VISIÓN COMPUTERIZADA APLICADA A LA MEDICINA

Roberto de Miguel López
Rubén Moreira López
Daniel Novillo Villarejo

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO

22 de junio de 2014

Supervisado por: Carlos Gregorio Rodríguez

Agradecimientos

Queríamos agradecer en primer lugar a Carlos Gregorio Rodríguez, director y máximo responsable de que este proyecto se haya llevado a cabo, gracias por la oportunidad que nos ha brindado de realizarlo y por toda la ayuda que nos ha proporcionado durante el curso para llegar a terminarlo.

Nuestro más sincero agradecimiento al equipo médico del Hospital de la Paz formado por: Álvaro González Miranda, Facultativo Especialista de Área; Ramón Varela, Médico Interno Residente de tercer año de Cirugía Plástica, Reparadora y Estética; Mónica Rubio Yanchuck y Carlos Brage Martín, ambos Médicos Internos Residentes de segundo año de Cirugía Plástica, Reparadora y Estética. Gracias por su colaboración aportando los diferentes casos de pacientes y sus fotografías, así como su asistencia a la reunión. Todo ello ha hecho posible este proyecto.

Por último, cabe agradecer a los pacientes que han permitido la toma de muestras para su estudio.

Gracias a todos por vuestra colaboración.

Índice

1. Introducción **1**

2. Introduction **3**

3. Metodología y recursos **5**

3.1. Metodología 5

3.1.1. Preparación y aprendizaje 5

3.1.2. Experimentación y problema real 6

3.1.3. Reatroalimentación con el tutor del proyecto y el equipo médico 6

3.2. Recursos 7

3.2.1. Herramientas 7

3.2.2. Librerías 7

3.2.3. Material 8

4. Conociendo OpenCV. **9**

4.1. Conceptos de visión artificial usados a lo largo del proyecto 9

4.1.1. Imagen: conceptos básicos 9

4.1.2. Blur 9

4.1.3. Template Matching 9

4.1.4. Canny 11

4.1.5. Ecualización de Histrograma 11

4.1.6. Threshold 13

4.1.7. K-Means 13

4.1.8. Filtro de Gabor 13

4.2. Primeros ejemplos 18
4.2.1. Problema concreto: Encontrar las diferencias entre dos imágenes 18
4.2.2. Problema concreto: ¿Dónde está Wally? 19

5. Análisis de heridas quirúrgicas **20**
5.1. Preprocesamiento de la imagen 20
5.2. Caminos distintos de estudio 24
5.2.1. K Means 24
5.2.2. Filtros de Gabor 37
5.3. Aportaciones de los miembros del grupo 41
5.3.1. Roberto de Miguel López 41
5.3.2. Rubén Moreira López 43
5.3.3. Daniel Novillo Villarejo 44

6. Discusión sobre los resultados **47**
6.1. Resultados 47
6.1.1. OpenCV 47
6.1.2. K-Means como prueba de concepto 47
6.1.3. Aplicación de detección de texturas a otros ámbitos 48
6.2. Consideraciones 48
6.2.1. Calidad de las muestras 48
6.2.2. Ecualización y tarjeta QP 49
6.2.3. Brillos, solución adoptada 49
6.2.4. Filtros de Gabor 50
6.2.5. Pruebas con el K-Means. 51
6.2.6. Problemas con la solución obtenida con K-Means 51

6.3. Trabajo futuro 53
6.3.1. OpenCV y la medicina 53
6.3.2. Machine Learning 53

7. Discussion of results **55**

7.1. Results 55
7.1.1. OpenCV 55
7.1.2. K-Means as concept testing 55
7.1.3. Texture detection application to other fields 55
7.2. Considerations 55
7.2.1. Quality of samples 56
7.2.2. Equalization and QP card 56
7.2.3. Glitters, adopted solution 56
7.2.4. Gabor filters 57
7.2.5. Test with K-Means. 58
7.2.6. Problems with the solution obtained by K-Means 58
7.3. Future work 59
7.3.1. OpenCV and medicine 60
7.3.2. Machine Learning 60

Índice de figuras

4.1. Tipos de Blur [9] 10

4.2. Ejemplo Gaussian Blur [9] 10

4.3. Ejemplo Template Matching [9] 11

4.4. Ejemplo Canny [9] 12

4.5. Ejemplo de imagen sin ecualizar [5] 13

4.6. Ejemplo de imagen ecualizada 13

4.7. Tipos de Threshold [7] 14

4.8. Ejemplo Threshold [7] 15

4.9. Funcionamiendo del K-Means durante dos iteraciones: (a) los centros (núcleos) se colocan al azar y cada punto se asigna a su centro más cercano; (b) los centros se mueven al centro de gravedad de sus puntos; (c) los puntos de datos se vuelven a asignar a sus centros más cercanos; (d) los centros se mueven de nuevo al centro de gravedad de sus puntos [9] 15

4.10. Parte real de la respuesta de impulso de un filtro de Gabor bidimensional [4] . . 16

4.11. Núcleos del filtro de Gabor con valores del parámetro de longitud de onda de 5, 10 y 15 píxeles, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: orientación 0 grados, desviación de fase 0 grados, relación de aspecto de 0.5, y ancho de banda 1 octava. 16

4.12. Núcleos del filtro de Gabor con valores del parámetro de orientación de 0, 45 y 90 grados, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10 píxeles, desplazamiento de fase 0 grados, relación de aspecto de 0.5, y ancho de banda 1 octava. 17

4.13. Núcleos del filtro de Gabor con valores de fase de 0, 180, -90 y 90 grados, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10 píxeles, orientación 0 grados, relación de aspecto de 0.5, y ancho de banda 1 octava. 17

4.14. Núcleos del filtro de Gabor con valores del parámetro de ancho de banda de 0.5, 1 y 2, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10 píxeles, orientación 0 grados, desplazamiento de fase 0 grados, y relación de aspecto de 0.5. 17

4.15. Núcleos del filtro de Gabor con valores del parámetro de relación de aspecto de 0.5 y 1, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10, orientación 0, desplazamiento de fase 0, y el ancho de banda 1. 18

4.16. Playa: Fotografía 1 18

4.17. Playa: Fotografía 2 18

4.18. Resta absoluta de la Figura 4.16 y Figura 4.17 19

4.19. Resta con contornos superpuesta con una de las imágenes originales 19

4.20. Escenario ¿Dónde está Wally? 20

4.21. Cara de Wally 20

4.22. Wally encontrado 20

5.1. Imagen original 22

5.2. Imagen ecualizada de la Figura 5.1 22

5.3. Imagen original 23

5.4. Herida recortada de la Figura 5.3 23

5.5. Primera aproximación al K-Means 25

5.6. Resultado tras aplicar el threshold (valores limpios). 26

5.7. Salida del algoritmo Canny combinado con la imagen original para apreciar las áreas 27

5.8. Salida K-Means 27

5.9. Canny de la salida del K-Means 28

5.10. K-Means con áreas coloreadas 29

5.11. K-Means puro con k=3 31

5.12. Porcentajes obtenidos con k=3 32

5.13. Fotografía tomada con flash 32

5.14. K-Means de la Figura 5.13 con k=4 33

5.15. K-Means de la Figura 5.13 con k=4 34

5.16. Detección de brillos. Aparecen marcados con una cruz todos los puntos de brillo 35

5.17. Zonas de brillos pintadas y rodeadas por un rectángulo 36

5.18. Imagen original 39

5.19. Imagen del núcleo con valores de los parámetros de la varianza de la función gaussiana 3, longitud de onda 80, orientación 37 y desplazamiento de fase 70 . . 39

5.20. Imagen con filtro de Gabor aplicado con valores de los parámetros de la varianza de la función gaussiana 3, longitud de onda 80, orientación 37 y desplazamiento de fase 70 39

5.21. Imagen de la Figura 5.20 elevada a potencia de dos 40

5.22. Imagen con filtro de Gabor aplicado con valores de los parámetros de la varianza de la función gaussiana 4, longitud de onda 69, orientación 45 y desplazamiento de fase 75 40

6.1. Ejemplo 1: Imagen original 47

6.2. Ejemplo 1 47

6.3. Ejemplo 2: Imagen original 48

6.4. Ejemplo 2 48

6.5. Ejemplo 3: Imagen original 48

6.6. Ejemplo 3 48

6.7. Fotografía tomada con flash 52

6.8. K-Means de la Figura 6.7 con k=3 52

Resumen

Este proyecto surge de una propuesta por parte de investigadores de la Universidad Complutense de Madrid en colaboración con personal médico del Hospital de la Paz para utilizar la potencia actual del procesamiento de imágenes a un tipo de heridas de características similares. En este caso, se ha diagnosticado la evolución de heridas quirúrgicas causadas en la extracción de zonas de capas superficiales de la piel, para su injerto en áreas con quemaduras.

El problema consiste en la evaluación de diferentes métodos y herramientas de visión computacional para establecer una base para futuras investigaciones y desarrollo de aplicaciones que sirvan para agilizar procesos médicos.

En esta memoria se muestran los diferentes caminos que se han seguido para la resolución de dicho problema, con los procedimientos seguidos en cada caso. Se muestran también los resultados obtenidos cuantitativamente y las imágenes resultantes, que servirán como fundamento para futuros estudios del tema.

Abstract

This project came up out of a proposal by the hand of investigators from the Universidad Complutense de Madrid in collaboration with medical staff from the Hospital de la Paz to use the current image processing power applied to a type of wounds with similar characteristics. In this case, it is the diagnostic of the evolution of surgical wounds caused due to the extraction of superficial dermis areas, so as to implant them over burnt skin.

The problem is to evaluate the different visual computer methods and tools to establish a base which would be useful for future investigations and development of applications that are used for speeding up medical procedures.

In this memory it has been illustrated the different paths that were followed to acquire the resolution of this problem, along with the procedures for each case. They are also shown the obtained results in a quantitatively way and the result images since they will serve as a basis for future studies connected to this topic.

Palabras clave

Visión computacional, OpenCV, herida quirúrgica, K-Means, filtros de Gabor, detección de brillos, zonas de interés, procesamiento de imágenes.

Keywords

Computer vision, OpenCV, surgical wound, K-Means, Gabor filters, glare detection, region of interest, image processing.

1. Introducción

En los últimos años se ha avanzado mucho en el campo de la informática. Aplicaciones que antes eran inviables debido a que el tiempo requerido para su ejecución era demasiado elevado, han dejado de serlo gracias al gran avance que ha habido en el hardware. Fruto de estos avances, ha sido el surgimiento de nuevo software capaz de utilizar de forma óptima estas nuevas tecnologías.

Un claro ejemplo sería el procesamiento de imágenes. En los últimos años se han desarrollado librerías muy potentes como `OpenCV`, una librería libre distribuida bajo la licencia `BSD` [3], de visión artificial, originalmente desarrollada por `Intel` en 1999. Esta librería pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esta memoria recoge el camino que hemos recorrido en el procesamiento de determinadas imágenes usando esta librería.

Problema

Un injerto de piel es la extracción y el trasplante de piel sana proveniente de una región del cuerpo (área fuente o sitio donante) a otra región (área receptora) donde la piel está lesionada. Las zonas fuente más comúnmente empleadas para injertos son la parte interna del muslo, pierna, glúteos, brazo y antebrazo.

En el caso de esta investigación, se presentan unos pacientes que han sufrido una quemadura y se les ha realizado un injerto de piel. No sólo la piel lesionada a la cual se ha trasplantado la piel sana es controlada por el equipo médico, sino que además la zona de la cual se extrae la piel (la zona de la herida quirúrgica) también debe estar bajo seguimiento.

Este tipo de heridas son ideales para automatizar su procesamiento, pues se trata de heridas muy similares unas a otras, ya que son heridas provocadas por instrumental médico en un entorno controlado, y su evolución será muy parecida.

La importancia de este hecho es fundamental; si todas las heridas tienen una evolución similar, podrán obtenerse datos que servirán de base para el aprendizaje automático; por ejemplo, una fotografía tomada 10 días después de la operación quirúrgica; con información cuantitativa de 40 % de piel sana, 20 % en proceso de epitelización y 20 % de tejido de granulación, se podría decir que está evolucionando correctamente por encontrarse dentro de unos márgenes establecidos.

Objetivo

Nuestro objetivo en este proyecto es intentar crear un prototipo de aplicación capaz de procesar la fotografía de la zona de la cual se ha extraído la piel sana para el injerto. De dicho

procesamiento se ha de obtener la mayor cantidad de información médicamente relevante: existencia de hematomas, restos de apósito hidrocoloide, tejido de granulación, piel epitelizada, etc.

Dado el escaso conocimiento de procesamiento de imágenes en entornos médicos que se tenía al inicio del proyecto y la dificultad del problema, éste se plantea como una prueba de concepto cuyo objetivo es obtener la mayor información posible, así como conocer la capacidad del procesamiento de imágenes aplicado a este tipo de heridas.

2. Introduction

In recent years a lot of progress has been made in the field of computing. Applications that were previously unfeasible due to the time required for implementation was too high, are no longer with the great progress that has been in the hardware. The result of these developments has been the emergence of new software capable of make optimal use of these new technologies.

A clear example is image processing. In recent years there have been very powerful libraries like `OpenCV`, a free library distributed under license `BSD` of computer vision, originally developed by `Intel` in 1999. This library seeks to provide a development environment easy to use and highly efficient. This report reflects the progress we have made in processing some images using this library.

Problem

A skin graft is the removal and transplantation of healthy skin from one area of the body (source area or donor site) to another area (recipient area) where the skin is injured. The source areas most commonly used for skin grafts are the inner thigh, leg, buttocks, arm and forearm.

In the case of this research, there are presented some patients who have been burnt and have undergone a skin graft. Not only injured skin in which has been transplanted healthy skin is controlled by the medical team, but also the area which the skin is extracted (the area of the surgical wound) must be also tracked.

This kind of wounds are ideal for automated processing, since it deals with almost alike wounds one another, as the wounds are caused by medical devices in a controlled environment, and its development will be very similar.

The significance of this fact is crucial; if all wounds have similar developments, data as a basis for automatic learning may be obtained; for example, a photograph taken 10 days after the surgical operation; with quantitative information of 40 % of healthy skin, 20 % in the process of epithelialization and 20 % of granulation tissue, it could be said that it is properly evolving within established margins.

Objective

Our goal in this project is the attempt to create a prototype application able to process the photograph of the area which has been extracted healthy skin for grafting. Out of such processing, it must be obtained as much medically relevant information: existence of hematomas, remains of hydrocolloid dressing, granulation tissue, epithelialized skin, etc.

Given the limited awareness of image processing in medical settings that was acknowledged in the very beginning of the project and the difficulty of the problem, it is presented as a proof of concept which its aim is to get as much information as possible, as well as to be familiar with the capacity of image processing applied to this type of injury.

3. Metodología y recursos

En esta sección serán descritos tanto los métodos usados para llevar a cabo la investigación como las herramientas y recursos que se han utilizado durante los mismos. El proceso ha consistido casi en su totalidad en el descubrimiento de caminos novedosos para resolver el problema, así como diferentes investigaciones alrededor de estos caminos para determinar su viabilidad, eficiencia y consumo de recursos, pudiendo de esta forma encontrar el más adecuado.

Para el desarrollo, se eligió el entorno de programación de código abierto `OpenCV` frente a otras alternativas como `Matlab`, dada la experiencia del tutor del proyecto con él y una velocidad de procesamiento mucho mayor.

Tras esta elección, se definieron varios caminos de acción en función de diferentes valores cuantitativos y cualitativos de las imágenes que constituían la base del estudio.

3.1. Metodología

El desarrollo de este proyecto puede dividirse en tres secciones, que a menudo se han repetido de forma cíclica retroalimentándose entre ellas, alcanzar la calidad esperada. Dichas secciones pueden separarse en: el aprendizaje de un método como solución a un problema determinado, el desarrollo de ese método y su ajuste para obtener los mejores resultados.

En las reuniones con el tutor del proyecto o con el equipo médico se validaban los resultados, de forma que si los requisitos no eran satisfechos de la forma esperada se produciría la retroalimentación en el proceso antes comentada.

3.1.1. Preparación y aprendizaje

El primer periodo de aprendizaje consistió en una investigación sobre las características que pueden ofrecer las diversas librerías de visión computacional con diferentes entornos de desarrollo, como son `OpenCV`, `Matlab`, etc. Tras un pequeño estudio, la librería candidata fue `OpenCV`, ya que al ser libre tiene una gran comunidad de desarrolladores a la espalda que permite que el código quede exento de muchos de los errores potenciales y dispone de bastante documentación a consultar, tanto en páginas oficiales como en pequeños dominios.

En esta fase se produjo la familiarización con el entorno de desarrollo, en el que se incluye `GitHub`, herramienta útil al tratarse de un proyecto entre varias personas.

Se comenzó con pequeños ejercicios supervisados, aplicaciones sencillas como *encontrar las diferencias* o posibles implementaciones de métodos clásicos propios de la visión computacional

como la localización de puntos de interés. Estos pequeños experimentos, junto con una serie de investigaciones a través de las propias `APIs` de `OpenCV`, hicieron que se tomase la decisión de usar como lenguaje de desarrollo `C++` en detrimento de `Python`, por motivos de experiencia previa.

3.1.2. Experimentación y problema real

Ante el problema de analizar cualitativa y cuantitativamente las imágenes de heridas quirúrgicas, surgieron dos aproximaciones que ofrecían caminos diferentes para llegar al mismo resultado, puesto que ambas eran capaces de aislar segmentos de la imagen en función de sus características, ya sea mediante el tratamiento de los colores o el tratamiento de texturas simples.

El primer método está basado en el algoritmo `K-Means`, en el cual se realiza una serie de preprocesados a la imagen en cuestión para luego utilizar los resultados como entrada a dicho algoritmo y aislar las diferentes zonas de color, las cuales suelen tener tres o cuatro áreas diferentes en función de qué tipo de imagen a tratar, su procedencia o su momento temporal.

El segundo método consiste en la estimación de parámetros y áreas en función de las texturas que muestra la imagen y otras características como la iluminación, para lograr sacar información útil de la imagen, así como limpiar impurezas como brillos o ruido.

3.1.3. Reatroalimentación con el tutor del proyecto y el equipo médico

Se ha procurado que la realización del proyecto fuera progresiva en el tiempo, de forma que se concertasen reuniones con el tutor, semanal o quincenalmente, en función de los avances realizados desde la reunión anterior.

En estas reuniones se presentaban las tareas realizadas, se discutían las soluciones propuestas por cada uno de los miembros del grupo y se sacaban las conclusiones pertinentes. Además, fijábamos los objetivos para la siguiente reunión, en los que podríamos incluir las modificaciones necesarias en las tareas presentadas ese día.

Cuando el proyecto estuvo lo suficientemente avanzado, se concertó una reunión con el equipo médico, cuyo objetivo principal fue el de la formación de los integrantes del grupo, para conocer qué información era necesario extraer y enfrentar, así, el problema de la manera más conveniente.

Asimismo, se mostraron y analizaron los avances a los médicos, de manera que se obtuvieron conclusiones importantes a la hora de enfocar el problema.

3.2. Recursos

Durante el desarrollo del proyecto, se usaron diferentes herramientas y librerías para poder cumplir los objetivos:

3.2.1. Herramientas

- NetBeans: Entorno de desarrollo integrado (IDE) libre, que admite diferentes lenguajes de programación, en este caso `C++`, gratuito y sin restricciones de uso. Tiene un sistema de plugins que facilita mucho las configuraciones. El proyecto debe incluir la compilación de librerías de OpenCV [8].
- Google Drive: Servicio de alojamiento de archivos. Usado principalmente para compartir las imágenes y fragmentos de código conflictivo

  entre los integrantes del equipo.
- GitHub: Entorno para alojar proyectos a través del sistema de control de versiones `Git`. Usado para alojar los diferentes pasos y caminos recorridos durante el desarrollo del proyecto.
- Moodle: Aplicación web que genera un entorno educativo virtual con sistema de gestión de cursos. En esta plataforma se creó un foro para dudas y puesta en común de avances, brainstorms, etc. junto con recursos como libros o investigaciones útiles para el proyecto.
- WriteLatex: Herramienta online para la creación y edición en `Latex`. Permite a varios usuarios editar el mismo documento a la vez.

3.2.2. Librerías

- OpenCV: Biblioteca (conjunto de librerías) libre de visión artificial originalmente desarrollada por Intel. Multiplataforma, contiene más de 500 funciones para el procesado de visión, reconocimiento de objetos, calibración de cámaras, visión estérea y visión robótica. Está enfocado a crear un entorno de desarrollo sencillo y altamente eficiente ya que su backend está programado en `C` y `C++` con una alta optimización. Además está adaptado a tecnologías actuales dadas sus capacidades en procesadores multinúcleo y GPUs.
- CvBlob: Librería de visión computacional para la detección de regiones conectadas en imágenes digitales binarias. Realiza un análisis de la conexión de los componentes y detección de características. Dadas sus similitudes con `OpenCV`, son compatibles entre sí.

3.2.3. Material

- Imágenes: El equipo médico asociado al proyecto proporcionó una serie de muestras de heridas quirúrgicas para el desarrollo del experimento. Estas muestras variaban desde la imagen única de un solo paciente hasta varias tomas del mismo paciente en distintos momentos temporales, registrando una progresión. Las imágenes poseían además una tarjeta QP y no pertenecían siempre a la misma parte del cuerpo, siendo las zonas más comunes el muslo o el brazo.
- Tarjeta QP: Junto con las imágenes, se administraba una tarjeta QP, que consiste en una superficie de papel plastificado en el que se muestra una paleta de colores. El principal objetivo de esta tarjeta es el de ayudar a la calibración de la cámara, aunque la propia empresa que suministra las tarjetas puede también suministrar software para corregir las desviaciones de color de una imagen junto con la tarjeta.

4. Conociendo OpenCV.

4.1. Conceptos de visión artificial usados a lo largo del proyecto

4.1.1. Imagen: conceptos básicos

En OpenCV, las imágenes están compuestas por una o varias matrices que guardan el valor del color de los píxeles que forman la imagen. Así pues, en la escala `RGB` (la más común), la matriz equivalente en openCV sería un objeto con tres matrices, cada una de las cuales guarda el valor Rojo (R) de un píxel, otra guarda el valor Verde (G) y otra guarda el valor Azul (B) y juntando estos tres componentes se puede sacar cualquier color de una imagen común.

Además, el valor de cada píxel puede variar dependiendo de la codificación de la imagen, siguiendo diferentes reglas: un píxel puede tener un valor de color que va de 0 a 255 en una codificación, mientras que en otra va de 0 a 1. La proporción es la misma pero varía su valor máximo.

4.1.2. Blur

El `Blurring`, también conocido como `Smoothing` es una operación de procesamiento de imágenes muy simple y muy utilizada. Hay muchas razones para usar esta técnica, entre ellas la de reducir el ruido de las imágenes o su resolución. OpenCV ofrece cinco tipos de blur diferentes (véase figura 4.1).

Se explicará el `Gaussian Blur`, que es el que se ha usado. Este es probablemente el más útil, aunque no el más rápido. Este filtrado se realiza mediante la convolución de cada punto en la matriz de entrada con un núcleo gaussiano y sumando luego para producir la matriz de salida. La intensidad con la que se genera el degradado de la imagen depende del tamaño del filtro, que es definido antes de realizar la operación (ejemplo en la Figura 4.2).

La convolución consiste en la superposición de las dos imágenes, la original y una versión trasladada e invertida del núcleo.

4.1.3. Template Matching

`Template Matching` es una técnica para la búsqueda de zonas de una imagen que se ajuste (o sea similar) a una imagen plantilla. Esta función no está basada en histogramas, si no que intenta hacer coincidir la plantilla *deslizándola* sobre la imagen usando diferentes métodos. En la Figura 4.3 tenemos como plantilla la imagen de una cara, que se irá deslizando sobre la imagen

Smooth type	Name	In place?	Nc	Depth of src	Depth of dst	Brief description
CV_BLUR	Simple blur	Yes	1,3	8u, 32f	8u, 32f	Sum over a `param1×param2` neighborhood with subsequent scaling by 1/(`param1×param2`).
CV_BLUR_NO_SCALE	Simple blur with no scaling	No	1	8u	16s (for 8u source) or 32f (for 32f source)	Sum over a `param1×param2` neighborhood.
CV_MEDIAN	Median blur	No	1,3	8u	8u	Find median over a `param1×param1` square neighborhood.
CV_GAUSSIAN	Gaussian blur	Yes	1,3	8u, 32f	8u (for 8u source) or 32f (for 32f source)	Sum over a `param1×param2` neighborhood.
CV_BILATERAL	Bilateral filter	No	1,3	8u	8u	Apply bilateral 3-by-3 filtering with color `sigma=param1` and a space `sigma=param2`.

Figura 4.1: Tipos de Blur [9]

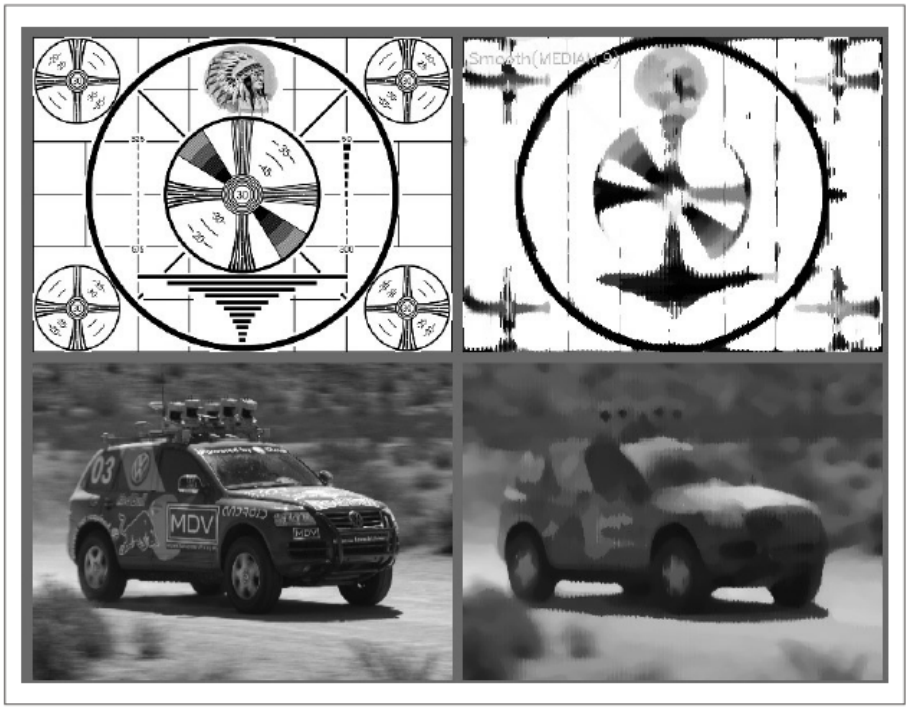


Figura 4.2: Ejemplo Gaussian Blur [9]

de entrada en busca de coincidencias.

Figura 4.3: Ejemplo Template Matching [9]

4.1.4. Canny

`Canny` es una técnica de detección de bordes. Esta función aplica un `filtro gaussiano` para reducir el ruido y después busca los bordes en vertical, horizontal y las dos diagonales. Dados los límites inferior y superior por el usuario, el algoritmo recorre píxel a píxel descartando aquellos con gradiente fuera de los límites definidos por el usuario, y considerando como borde a aquéllos en este rango sólo si están conectados a otro píxel en estas mismas condiciones (ejemplo en la Figura 4.4).

4.1.5. Ecualización de Histrograma

La ecualización de histograma es un método en el procesamiento de imágenes de contraste de ajuste utilizando el histograma de la propia imagen. Este método por lo general aumenta el contraste global de muchas imágenes, especialmente cuando los datos utilizables de la imagen están representados por valores de contraste próximos. A través de este ajuste, las intensidades se pueden distribuir mejor en el histograma. Esto permite que las áreas de menor contraste local obtengan un contraste más alto. La ecualización del histograma logra esto mediante la difusión de manera efectiva de los valores de intensidad más frecuentes.

El método es útil en imágenes con fondos y primeros planos que son a la vez brillantes y

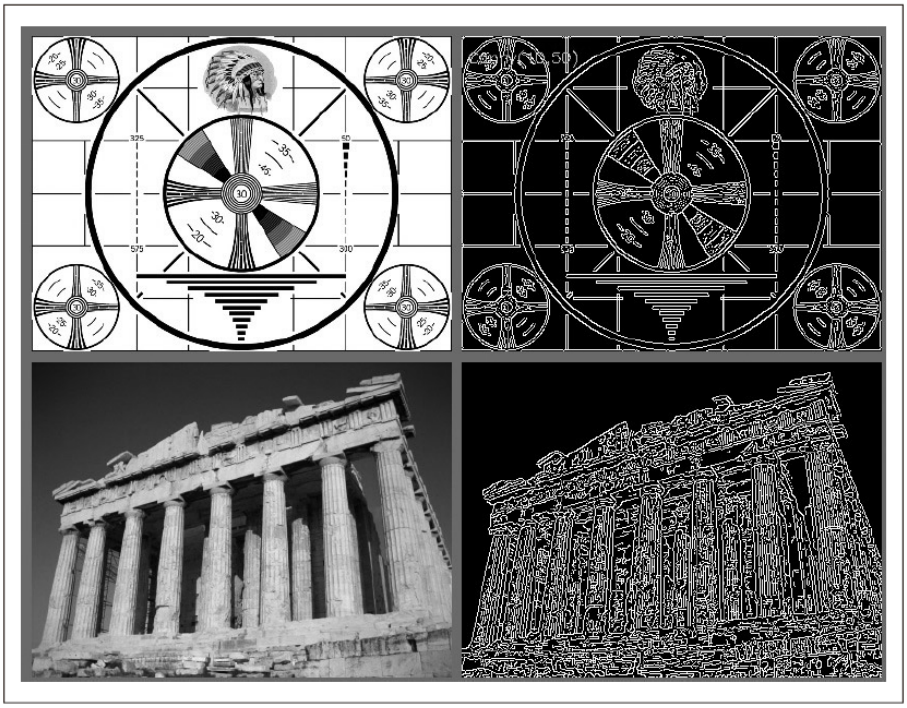


Figura 4.4: Ejemplo Canny [9]

oscuros. En particular, puede mejorar el detalle en las fotografías que se encuentran sobre o subexpuestas.

Una ventaja clave del método es que es una técnica bastante sencilla y que es posible invertir. Así que en teoría, si la función de la ecualización del histograma es conocida, entonces el histograma original puede ser recuperado. El cálculo no es computacionalmente intensivo, es decir, no utiliza muchos recursos.

Una desventaja del método es que es bastante indiscriminado: puede aumentar el contraste del ruido de fondo, mientras que disminuye el de la señal utilizable. También la ecualización del histograma puede producir efectos no deseados (como visible gradiente de imagen) cuando se aplica a imágenes con baja profundidad de color.

La ecualización implica el mapeo de una distribución (el histograma dado) a otra más amplia y más uniforme, para que los valores de intensidad se expandan en todo el rango.

Se puede utilizar este método en imágenes en color mediante la aplicación del mismo procedimiento separado para el rojo, verde y azul de los componentes `RGB` de la imagen. Sin embargo, puede producir cambios drásticos en la imagen de equilibrio de color ya que la distribución relativa de los canales de color cambia como resultado de la aplicación del algoritmo.

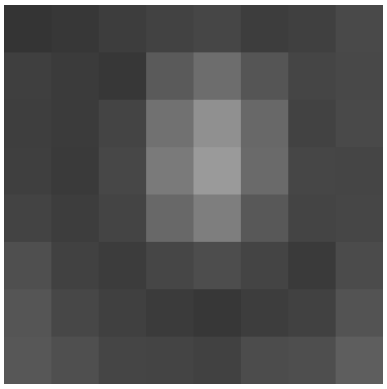


Figura 4.5: Ejemplo de imagen sin ecualizar [5]

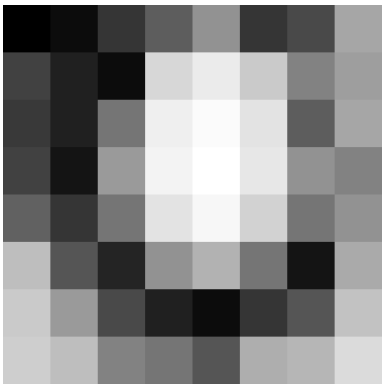


Figura 4.6: Ejemplo de imagen ecualizada

4.1.6. Threshold

El `Threshold` es el método más simple de segmentación. La idea básica es que dada una imagen, junto con un umbral, se recorre cada píxel, y dependiendo de si está por debajo o por encima del umbral, se toma una decisión en función del tipo de `Threshold` usado (Véase en la Figura 4.7).

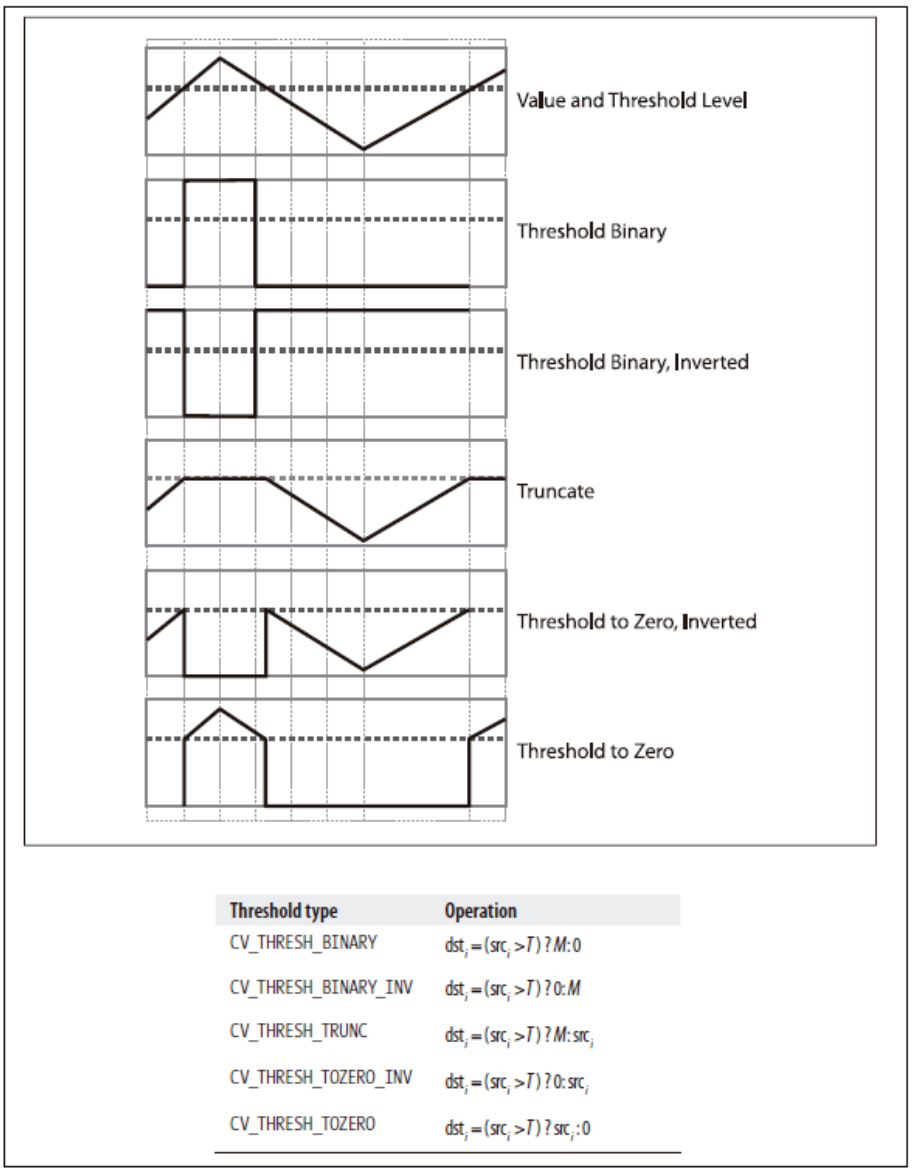
En nuestro caso, usábamos el `Threshold Binario` para descartar los píxeles fuera de este umbral. Si la intensidad píxel src(x, y) es mayor que el valor umbral, entonces la nueva intensidad de los píxeles se establece en un valor máximo fijado. De lo contrario, los píxeles se establecen en 0. Se puede ver un ejemplo en la Figura 4.8.

4.1.7. K-Means

El `K-Means` en un algortimo de agrupamiento que tiene como objetivo el reparto de n observaciones o datos en subconjuntos o grupos de k elementos, obteniéndose previamente mediante diferentes algoritmos, unos de ellos aleatorios y otros consecuentes con la imagen, unos núcleos que se usarán para medir la distancia entre un núcleo y cualquier punto de la muestra y de esta forma determinar cual es el núcleo más cercano a cada uno de los puntos. Esto provocará que el punto cuya distancia sea menor al núcleo n pertenezca al grupo de dicho núcleo. Se puede ver un ejemplo gráfico de su funcionamiento en la Figura 4.9.

4.1.8. Filtro de Gabor

El `filtro de Gabor` [12] es un filtro lineal cuya respuesta de impulso es una función sinusoidal multiplicada por una función gaussiana. El filtrado de una imagen con `funciones de Gabor` está relacionado con los procesos en la corteza visual. Concretamente, son un buen modelo para



Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T) \,?\, M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) \,?\, 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) \,?\, M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) \,?\, 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) \,?\, src_i : 0$

Figura 4.7: Tipos de Threshold [7]

los campos receptivos de las células simples de la corteza cerebral si se supone que éstas poseen un comportamiento lineal.

Por otra parte, se conoce a partir de estudios psicológicos que ciertas tareas, por ejemplo, detección de la cara, dependen principalmente de la información en las frecuencias ópticas intermedias. Esto puede ser porque los filtros de alta frecuencia sólo ven una pequeña imagen de la región y son filtros, por lo tanto, ruidosos y relativamente poco informativos; y los filtros de baja frecuencia actúan sobre una gran región y responden de manera desproporcionada a cambios lentos debidos a iluminación.

Figura 4.8: Ejemplo Threshold [7]

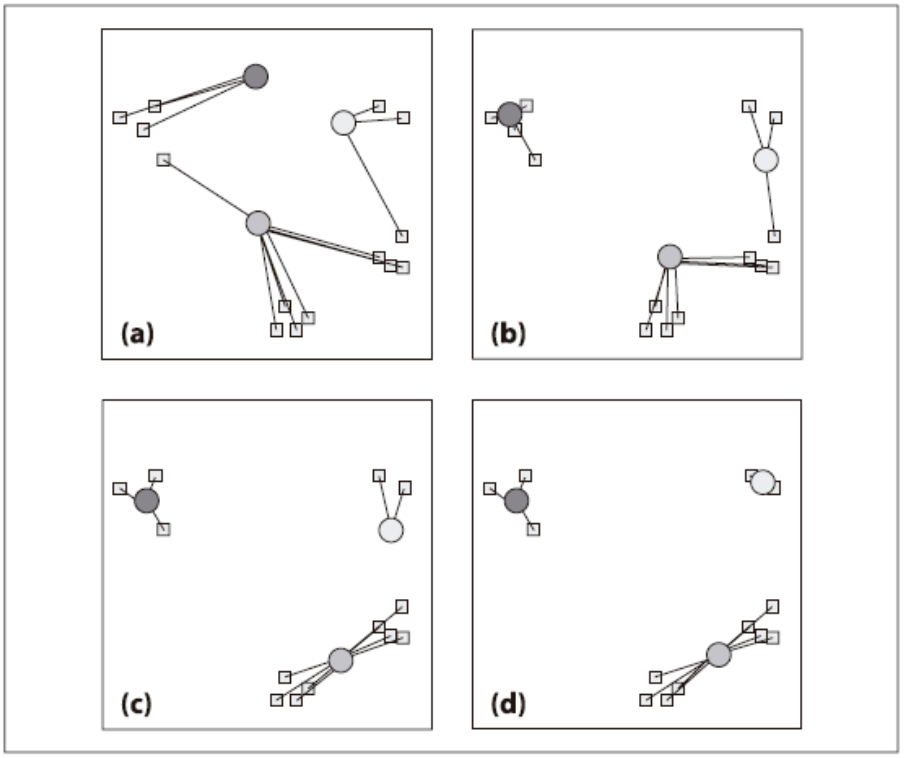


Figura 4.9: Funcionamiendo del K-Means durante dos iteraciones: (a) los centros (núcleos) se colocan al azar y cada punto se asigna a su centro más cercano; (b) los centros se mueven al centro de gravedad de sus puntos; (c) los puntos de datos se vuelven a asignar a sus centros más cercanos; (d) los centros se mueven de nuevo al centro de gravedad de sus puntos [9]

Además, los `filtros de Gabor` se han empleado en el procesamiento digital de imágenes,donde se han mostrado eficientes a la hora de realizar diferentes tareas, tales como segmentación de texturas, compresión, etc. Este primer hecho sirvió de inspiración para el tratamiento de las imágenes en el desarrollo de la aplicación.

Cada parámetro de la `función de Gabor` aplica cambios al núcleo resultante de forma diferente y única. Este núcleo se utiliza después para obtener la imagen final por medio de la

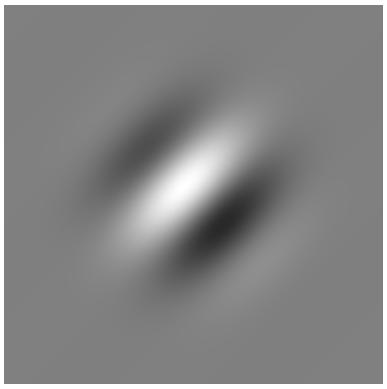


Figura 4.10: Parte real de la respuesta de impulso de un filtro de Gabor bidimensional [4]

convolución [11].

Longitud de onda (λ): Esta es la longitud de onda del factor coseno del núcleo del `filtro de Gabor`. Su valor se especifica en píxeles. Con el fin de prevenir la aparición de efectos no deseados en los bordes de la imagen, el valor de longitud de onda debe ser menor que un quinto del tamaño de la imagen original.

Figura 4.11: Núcleos del filtro de Gabor con valores del parámetro de longitud de onda de 5, 10 y 15 píxeles, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: orientación 0 grados, desviación de fase 0 grados, relación de aspecto de 0.5, y ancho de banda 1 octava.

Orientación (θ): Este parámetro especifica la orientación de la normal a las franjas paralelas de una `función Gabor`. Su valor se puede especificar en grados entre 0 y 360, pero en la aplicación se fija entre 0 y 180 dado que es simétrico.

Desviación de fase (ψ): El argumento del factor coseno de la `función Gabor` fase se especifica en grados. Los valores válidos son números reales entre -180 y 180, aunque en la aplicación se representan con valores entre 0 y 360, siendo 180 el valor real 0. Los valores reales 0 y 180 corresponden a funciones centrosimétricas, mientras que -90 y 90 corresponden a las funciones de antisimétricas. Todos los otros casos corresponden a funciones asimétricas.

Figura 4.12: Núcleos del filtro de Gabor con valores del parámetro de orientación de 0, 45 y 90 grados, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10 píxeles, desplazamiento de fase 0 grados, relación de aspecto de 0.5, y ancho de banda 1 octava.

Figura 4.13: Núcleos del filtro de Gabor con valores de fase de 0, 180, -90 y 90 grados, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10 píxeles, orientación 0 grados, relación de aspecto de 0.5, y ancho de banda 1 octava.

Varianza de la función gaussiana(σ): Este parámetro controla la propagación o radio del núcleo. Esta estrechamente relacionado con el ancho de banda de frecuencia espacial de un `filtro de Gabor`.

Figura 4.14: Núcleos del filtro de Gabor con valores del parámetro de ancho de banda de 0.5, 1 y 2, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10 píxeles, orientación 0 grados, desplazamiento de fase 0 grados, y relación de aspecto de 0.5.

Relación de aspecto (γ): Este parámetro, llamado más precisamente la relación de aspecto espacial, especifica la elipticidad del soporte de la `función de Gabor`. Para valor uno, el soporte es circular. Para valores menores que uno, el soporte es alargado en la orientación de las rayas paralelas de la función. El valor utilizado en la aplicación no es variable y siempre está asignado a uno; ya que se quería que fuera circular.

Figura 4.15: Núcleos del filtro de Gabor con valores del parámetro de relación de aspecto de 0.5 y 1, de izquierda a derecha, respectivamente. Los valores de los otros parámetros son los siguientes: longitud de onda 10, orientación 0, desplazamiento de fase 0, y el ancho de banda 1.

4.2. Primeros ejemplos

Esta fase de preparación se planteó como una serie de metas a alcanzar cada semana. De esta forma, el primer objetivo fue el de la instalación de la librería, y familiarización con `GitHub`, la herramienta que usaríamos para alojar nuestro proyecto utilizando el sistema de control de versiones `Git`.

Completada la primera meta, se propusieron problemas concretos como método de aprendizaje del uso de la librería `OpenCV`.

4.2.1. Problema concreto: Encontrar las diferencias entre dos imágenes

Dadas dos imágenes casi idénticas, en la que varían pequeños detalles, se deben encontrar las diferencias entre ellas.

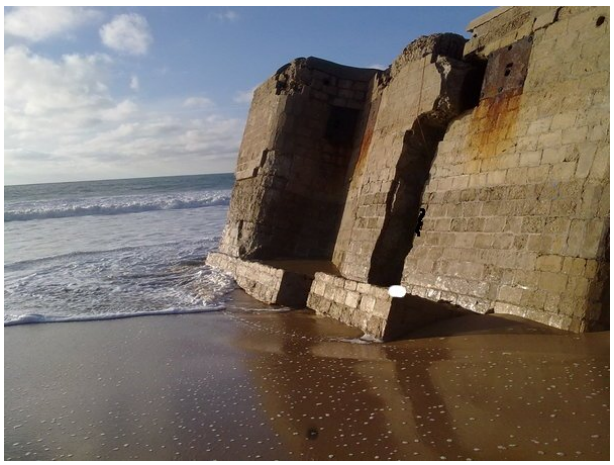


Figura 4.16: Playa: Fotografía 1

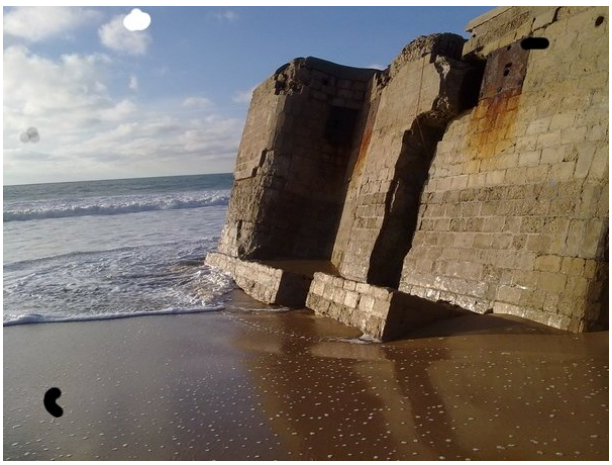


Figura 4.17: Playa: Fotografía 2

Las imágenes consisten en matrices, originalmente tridimensionales, que son transferidas a una

escala de grises (bidimensionales), dado que así se simplifica el problema. Utilizando imágenes en escala de grises se obtiene una mejora en la eficiencia del algoritmo para agilizar la muestra en tiempo real y poder ajustar de forma óptima los parámetros. Una vez obtenidas las imágenes en este formato, se realiza una resta absoluta para conocer aquellos sectores que sean diferentes entre ambas imágenes, aplicando después un `Blur` para eliminar el posible ruido que se haya generado ya que éste podría comprometer el resultado.

Por simple motivo de estética y para poder apreciar estas diferencias, se ha aplicado el algoritmo de detección de bordes `Canny` a la Figura 4.18. , y se ha superpuesto con una de las dos imágenes originales, obteniendo la Figura 4.19, en la cual se observan perfectamente todas estas diferencias.

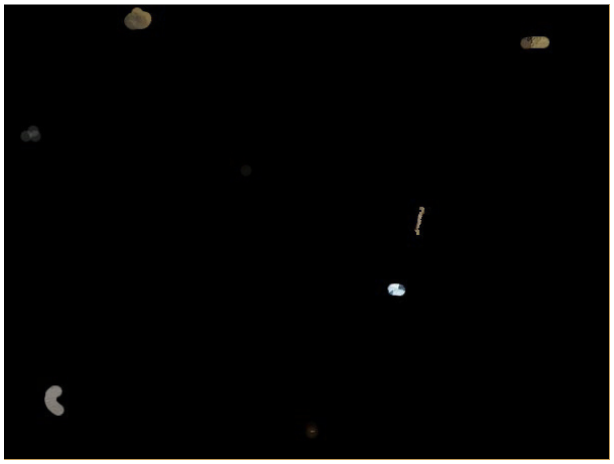


Figura 4.18: Resta absoluta de la Figura 4.16 y Figura 4.17

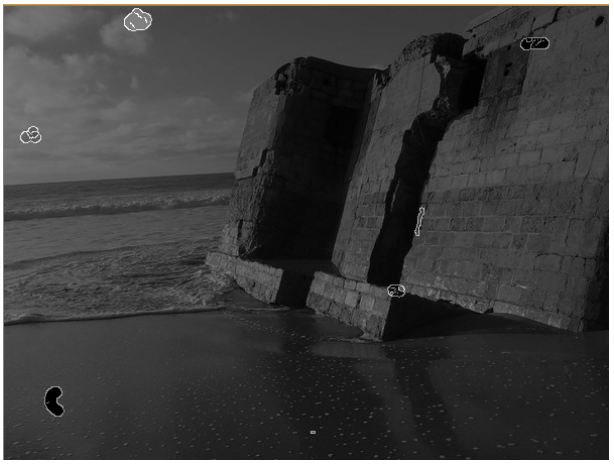


Figura 4.19: Resta con contornos superpuesta con una de las imágenes originales

4.2.2. Problema concreto: ¿Dónde está Wally?

Dada una imagen de un escenario: *¿Dónde está Wally?* éste debe ser localizado.

Este problema es ciertamente muy complejo, y se propuso para experimentar con el `MatchTemplate`. Para usar este método es necesario proporcionarle dos imágenes: el escenario y la cara de Wally (el objetivo de la búsqueda). El método usado no es la solución óptima, no va a funcionar en el 100 % de los casos, porque no siempre Wally aparece en las mismas condiciones o poses.

Con la imagen de la cara de Wally recortada del escenario, o bien usando una muy similar, se encuentra correctamente como se observa en la Figura 4.22.

Figura 4.20: Escenario ¿Dónde está Wally?

Figura 4.21: Cara de Wally

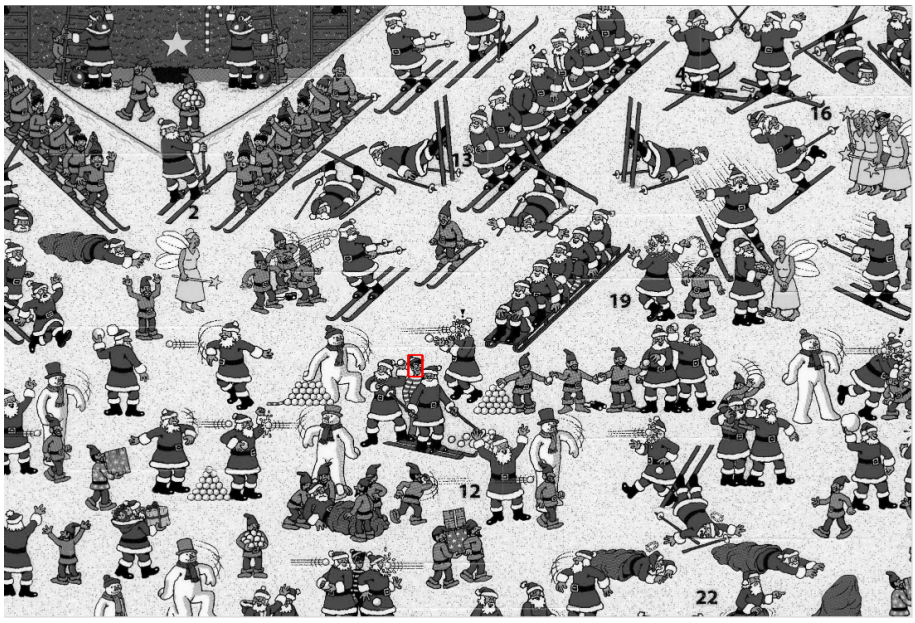


Figura 4.22: Wally encontrado

5. Análisis de heridas quirúrgicas

5.1. Preprocesamiento de la imagen

El primer objetivo de la fase de resolución fue la preparación de las imágenes para obtener otras más adecuadas para la aplicación, mejorando o eliminando ciertas características de la

mismas que posibilitara efectuar operaciones sobre ellas.

La finalidad del preprocesamiento de las imágenes fue:

- Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles colindantes.
- Eliminar ruido: descartar aquellos píxeles cuyo nivel de intensidad sea muy diferente al de sus adyacentes y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.
- Recortar la imagen: delimitar de forma efectiva la zona de aplicación de las técnicas.

Obtener colores de la tarjeta QP

Todas las fotografías están tomadas junto con una tarjeta de color QP. Dicha tarjeta tiene en sus cuatro esquinas un mismo color, sin embargo, por causas de iluminación o flash, no tienen en ningún caso los mismos valores RGB, por bastante diferencia.

Para poder usar la tarjeta QP para la ecualización de la imagen, sería necesario saber estos cuatro valores e implementar un algoritmo para obtener el color real. Para conocer el valor de estos cuatro colores, se decidió pedir al usuario que interactuase marcándolos con el cursor.

Sin embargo, el color de un píxel en concreto no es algo significativo, ya que en una misma zona de, aparentemente, un mismo color, se obtienen valores muy distintos. Se optimizó calculando la media de los píxeles circundantes, para obtener un valor lo más aproximado al real.

Ecualización de la imagen

Para poder analizar la imagen de forma efectiva era necesario que los colores se ajustasen, debido a que la luz de interior como el flash de la imagen pueden ser problemáticos. Se propuso utilizar una técnica de ajuste de contraste basada en la ecualización del histograma para solucionarlo, además de técnicas `SIFT` y de `Bundle Adjustment` para tratar de localizar la tarjeta QP usada en la toma de fotos y realizar un ajuste de color comparándola con una imagen limpia de otra tarjeta idéntica. Esto último trajo problemas, ya que no en todas las imágenes la tarjeta se visionaba de la misma manera, solía estar cortada o manchada, por lo que ante la posibilidad de invertir demasiados recursos en esto, se optó por usar la ecualización de histograma para limpiar un poco los colores y que se unificasen.

En resumen, los pasos que seguimos para ecualizar la imagen fueron:

- Cambiar el color de la imagen de `RGB` a formato `YCbCr` utilizando `cvtColor`. Esta conversión fue necesaria porque únicamente se tenía que ecualizar el componente de intensidad. La primera componente en `YCbCr` representa dicha magnitud, mientras que en `RGB` las tres componentes son de color.
- Separar la imagen en canales mediante `split`. Esta función divide cada canal de la matriz de la imagen en canales separados y los almacena en un vector uno a uno.
- Ecualizar el histograma en el primer canal (Y) utilizando `equalizeHist`. Los otros dos canales permanecen intactos ya que son las componentes de color.
- Fusionar los tres canales en una sola imagen `YCrCb` utilizando `merge`. Esta función realiza la operación inversa de la función `split`. Mediante el vector de canales crea una sola matriz multicanal.
- Cambiar el color de la imagen de `YCrCb` a formato `RGB`, para mostrar la imagen correctamente.

Esta forma de ecualizar la imagen resalta las diferentes partes de la herida aumentando el contraste. Por otro lado, el color de la imagen se muestra de forma irreal (véase Figura 5.2).

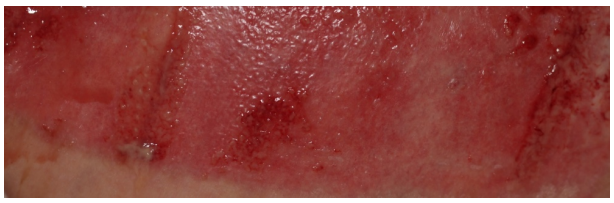


Figura 5.1: Imagen original

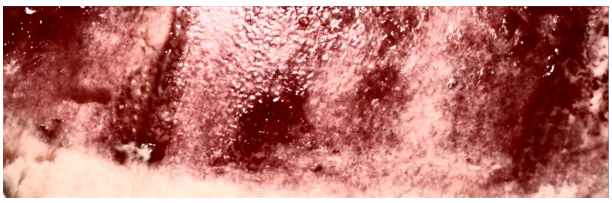


Figura 5.2: Imagen ecualizada de la Figura 5.1

Separar imagen a analizar de la tarjeta QP

Se decidió que el método más rápido y eficaz de separar la tarjeta QP del resto de la imagen era interactuando con el usuario. En una primera versión solo se solicitaba que el usuario pinchara sobre la esquina inferior de la tarjeta, ya que en las fotos de las que se disponía, la tarjeta estaba siempre colocada en la parte superior de la imagen. Una vez el usuario hubiese pinchado en ese punto se recortaba la imagen.

Sin embargo, con la llegada de nuevas muestras en las que la tarjeta no siempre estaba arriba, y en las que también salían zonas que carecían de interés (y que podían estropear por completo el procesamiento de la imagen) se buscó un método más adecuado.

Para ello se ideó un algoritmo de recorte basado en la extracción de regiones de interés o `ROI (Region of Interest)`. Este algoritmo crea una rectángulo mediante la selección de las esquinas por el usuario y crea una nueva imagen de esa región.

Con esta nueva imagen se tiene una zona más delimitada de la herida, aunque no totalmente aislada debido a que se extrae una zona con forma rectangular.

Figura 5.3: Imagen original

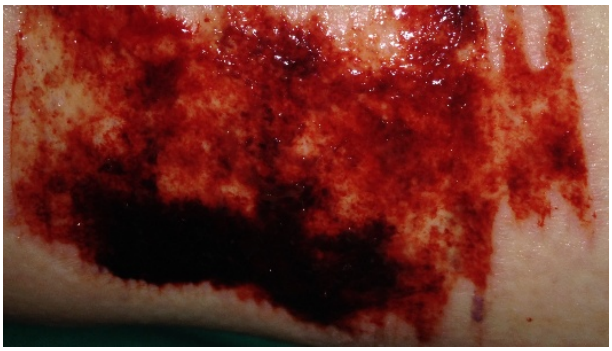


Figura 5.4: Herida recortada de la Figura 5.3

5.2. Caminos distintos de estudio

Ante la magnitud del problema, se plantearon dos caminos diferentes para afrontarlo. El primero consistía en una aproximación mediante el algoritmo de agrupación por centroides llamado `K-Means`, de esta forma se juntan todos aquellos puntos similares en color unos con otros, creando K áreas diferentes. El segundo procedimiento se basaba en la posibilidad de la diferenciación de las texturas en la imagen para poder discriminar unas regiones de otras mediante la aplicación de filtros de detección de bordes.

5.2.1. K Means

Al observar las imágenes, una de las primeras impresiones que se tuvieron fue la cantidad de sangre y texturas enrojecidas que se observaban. Dichas observaciones, dependiendo de la tonalidad de este rojo, se diagnosticaban de una forma u otra. Esto se podía utilizar para diferenciar y clasificar las distintas zonas de la herida. Para lograrlo era necesario aplicar una serie de técnicas de ecualización para que los colores fuesen tan realistas como fuese posible (Véase Figura 5.2), dado que las técnicas de calibración de imágenes eran muy costosas computacionalmente.

Una vez solucionado el problema del preprocesado, se llevaron a cabo investigaciones sobre diferentes algoritmos para obtener clasificaciones en función de patrones de color, y tras estudiar varios, se llegó a la conclusión de que el mejor para la tarea actual era el K-Means.

Tras una serie de ajustes, estudios y pruebas intensivas, se llegó a unos primeros resultados con este algoritmo:

Esta imagen está tomada con un K por valor de 8 núcleos y un epsilon de 0.5. Se denotan claramente las zonas con texturas y las zonas de piel más lisa. Si se ajustaba la epsilon, las zonas se iban lentamente distinguiendo hasta llegar a la clara diferenciación que se ve en la imagen.

Como zona principal se encuentra la piel epitelizada, diferenciando así lo que es piel en proceso de curación de lo que no lo es, ya sea porque es piel completamente curada (como ocurre en la parte inferior de la figura), o zonas más comprometidas o que aún no han empezado la curación (que corresponden a zonas de costras, hematomas o granulocitos).

Una primera aproximación sugirió que se podría aplicar una técnica de aprendizaje computerizado guiado para, en imágenes subsecuentes, localizar las diferentes áreas. Esta idea fue descartada ante las pocas muestras de las que se disponía, por lo que se buscaron otras técnicas para aprovechar el resultado del k-means. Para que los resultados fuesen más característicos, se procedió a limpiar los colores de la imagen con un sencillo algoritmo para así poder tener en blanco las zonas importantes a estudiar y poder medir estas áreas, cuantificarlas y diferenciarlas, ya que la codificación de la imagen obtenida no era la misma que la que se podía utilizar. El algoritmo en cuestión es el de `Threshold`, cuya función consiste en separar los valores de una

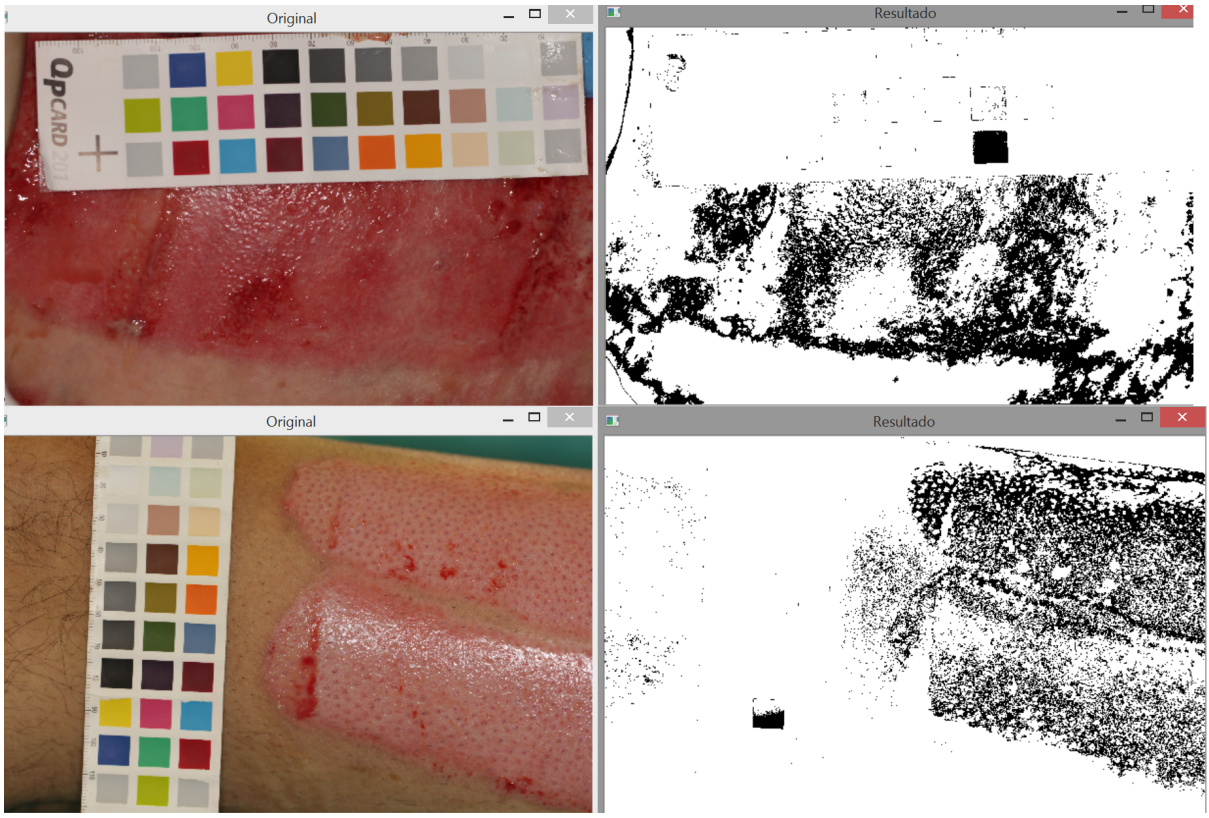


Figura 5.5: Primera aproximación al K-Means

imagen en función de si sobrepasa o no un valor. En el caso de la imagen, todo aquel valor que no era blanco absoluto, se ponía a negro (véase Figura 5.6).

Tras estos primeros pasos, se realizaron intentos para delimitar las áreas objetivo, puesto que las zonas en blanco eran candidatas de estudio para extraer información adicional a posteriori. Se estudiaron diferentes caminos de bordeado, como el método de `Harris`, el de `Canny` o métodos de detección de características. Pero de todos ellos, `Canny` fue el que conllevó mejores resultados con un menor coste computacional. Esto funciona porque no se usaba la salida estándar del algorítmo `K-Means`:

El `K-Means` devuelve una serie de valores que corresponden a cada uno de los núcleos de la imagen, además de la matriz de los centros de los clusters. Si se obvia el número de núcleos dado y se efectúa un threshold sobre la matriz salida sin repartirlo de forma homogénea en la escala de colores, se obtiene un resultado como el la figura 5.6. Esto permite separar todos y cada uno de los clusters del que ha sido designado como base (en negro), el cual, en el caso de las imágenes, implica la separación de la imagen en dos áreas completamente distintas y bien diferenciadas, como son para el caso la zona en recuperación y la zona inversa a ésta. El threshold es una operación necesaria porque los valores van desde 0 hasta números infinitesimalmente cercanos a

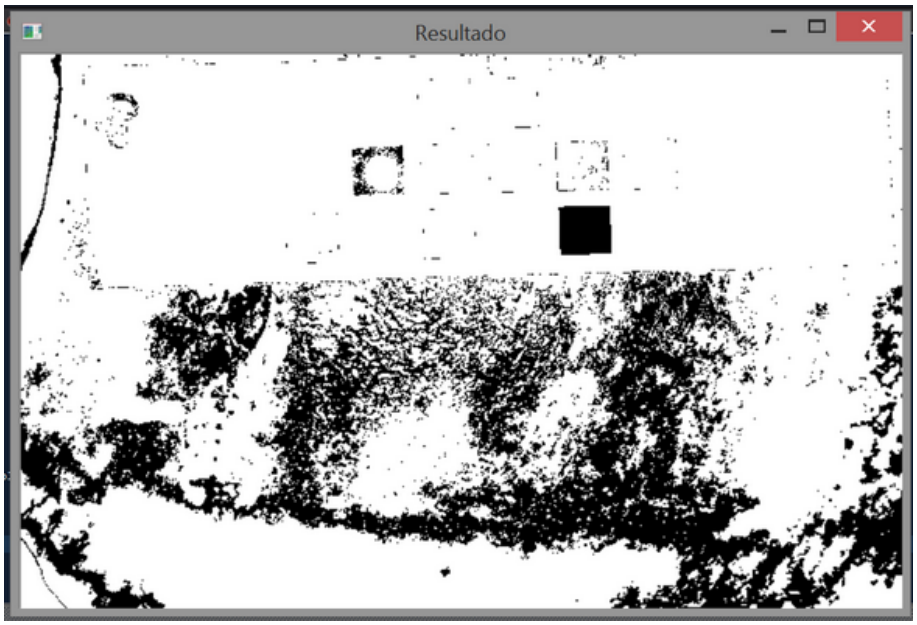


Figura 5.6: Resultado tras aplicar el threshold (valores limpios).

1, debido a la codificación con la que el algoritmo devuelve la matriz de datos como salida. El `Threshold` permite así mantener todos aquellos valores que sean cero en cero, mientras que todo aquel que no corresponda a este valor se eleva a 255 para que el formato final de la imagen sea visible y quedando la imagen en blanco y negro de la figura 5.6. Al tener dos áreas perfectamente diferenciadas con colores opuestos, los algoritmos de detección de bordes trabajaban con una precisión muy alta, obteniendo una salida valiosa que permitía localizar las zonas consideradas de interés en las imágenes (Figura 5.7).

Además el algoritmo se podía ajustar en función de los núcleos, y aunque no era la forma más adecuada de usar la salida del algoritmo, daban ciertos resultados en algunas de las imágenes, como por ejemplo las posibles peores zonas de la herida (esto con un número de núcleos aproximadamente de 10) o las zonas de la herida que se sacan en la Figura 5.7 (bajo el valor de 6 núcleos).

Se inició una pequeña investigación, dado que en cada iteración, debido al origen pseudo-aleatorio de los núcleos, los resultados variaban ligeramente por pequeños detalles (y a veces no tan pequeños) pero conservaban la idea de ese valor de K. De esta forma, se creó un pequeño algoritmo que ejecutaba el `K-Means` bajo el mismo valor de K y registraba las variaciones de las áreas para luego calcular la media y mostrarla, enriqueciendo así la información objetivo. La naturaleza aleatoria del algoritmo provocaba que en sucesivas ejecuciones los resultados se fuesen corrompiendo cada vez más hasta no formar ningún resultado útil (formas aleatorias que nada tenían que ver con lo que se observaba en las muestras), por lo que este camino quedó rápidamente descartado.

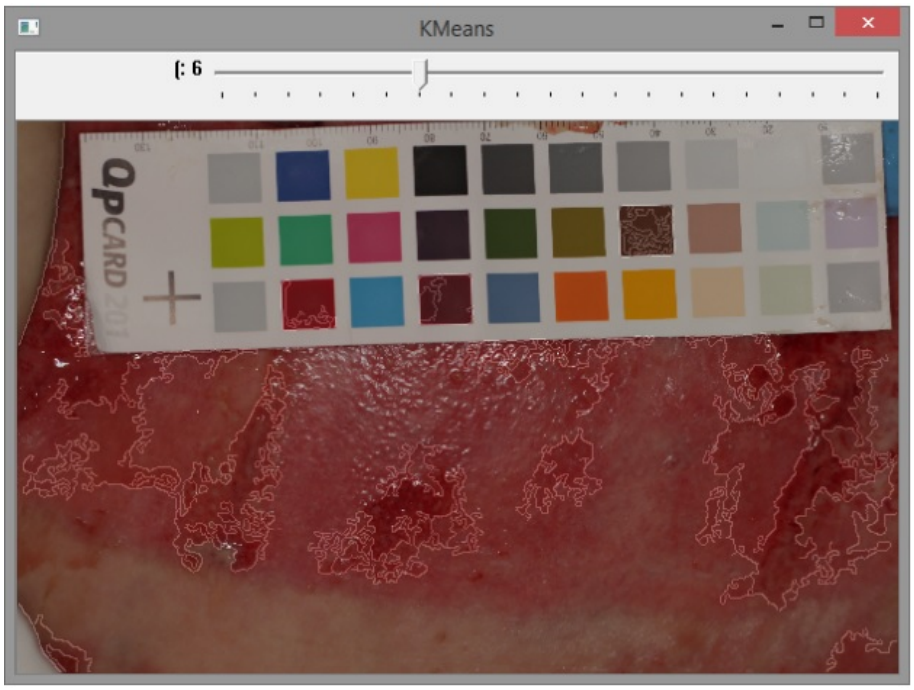


Figura 5.7: Salida del algoritmo Canny combinado con la imagen original para apreciar las áreas

5.2.1.1. Aprovechamiento del K-Means

La salida del `K-Means` usado muestra las diferentes zonas encontradas de la herida, pero no se pueden obtener conclusiones cuantitativas ni visuales.

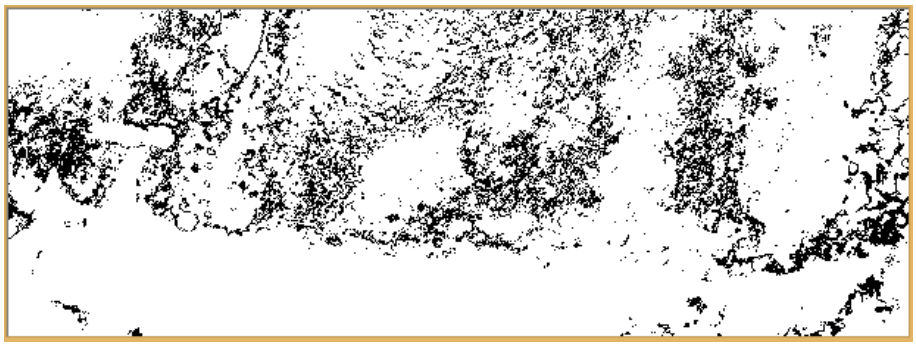


Figura 5.8: Salida K-Means

Una vez pasado este punto, los esfuerzos se centraron en encontrar las diferentes áreas, y colorearlas. Para ello se delimitaron los bordes de la salida del `K-Means`, aplicando después un `Dilate` para descartar aquellos bordes de no más de un punto.

Sin embargo, con los bordes no se obtiene ni mucho menos el resultado deseado, como podemos ver en la Figura 5.9.

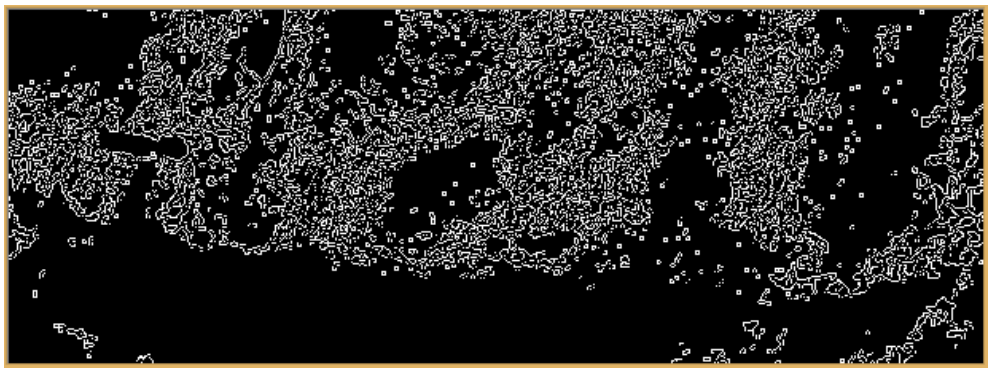


Figura 5.9: Canny de la salida del K-Means

Obteniendo los contornos de la salida del `K-Means`, ya se podrían delimitar las áreas. Este paso es muy importante, ya que calculando las áreas, ya podemos cuantificar que porcentaje de la herida parece que es piel sana, que porcentaje es tejido de granulación... Además, se quiso dar un aspecto visual coloreando estas áreas en diferentes tonos.

Mediante la función de `OpenCV findContours` se obtienen los contornos de la imagen, para luego poder obtener los momentos, los centros de masa, y así calcular las áreas.

Se obtendría esta información sobre las áreas, datos poco útiles a priori dado el número de diferentes áreas encontradas, aunque el resultado visual sea mejor, como se muestra en la Figura 5.10

```
Info: Area and Contour Length \cite{moments}

 * Contour[10] - Area (M_00) = 106 - Area OpenCV: 106 - Length: 58.1421
 * Contour[11] - Area (M_00) = 232.5 - Area OpenCV: 232.5 - Length: 100.87
 * Contour[12] - Area (M_00) = 126.5 - Area OpenCV: 126.5 - Length: 51.8995
 * Contour[22] - Area (M_00) = 56.5 - Area OpenCV: 56.5 - Length: 54.7279
 * Contour[30] - Area (M_00) = 76 - Area OpenCV: 76 - Length: 46.1421
 * Contour[32] - Area (M_00) = 53.5 - Area OpenCV: 53.5 - Length: 33.5563
 * Contour[43] - Area (M_00) = 85 - Area OpenCV: 85 - Length: 61.3137
 * Contour[47] - Area (M_00) = 294.5 - Area OpenCV: 294.5 - Length: 123.012
 * Contour[59] - Area (M_00) = 139 - Area OpenCV: 139 - Length: 82.1421
 * Contour[76] - Area (M_00) = 109.5 - Area OpenCV: 109.5 - Length: 53.5563
 * Contour[81] - Area (M_00) = 335 - Area OpenCV: 335 - Length: 109.113
 * Contour[89] - Area (M_00) = 15658.5 - Area OpenCV: 15658.5 - Length: 2398.4
 * Contour[99] - Area (M_00) = 45.5 - Area OpenCV: 45.5 - Length: 39.5563
```

```
* Contour[131] - Area (M_00) = 96.5 - Area OpenCV: 96.5 - Length: 66.0416
* Contour[190] - Area (M_00) = 64.5 - Area OpenCV: 64.5 - Length: 43.5563
* Contour[213] - Area (M_00) = 112 - Area OpenCV: 112 - Length: 59.799
* Contour[239] - Area (M_00) = 133 - Area OpenCV: 133 - Length: 75.4558
* Contour[299] - Area (M_00) = 57.5 - Area OpenCV: 57.5 - Length: 41.5563
* Contour[309] - Area (M_00) = 51217 - Area OpenCV: 51217 - Length: 2973.93
* Contour[351] - Area (M_00) = 352 - Area OpenCV: 352 - Length: 111.598
* Contour[431] - Area (M_00) = 87 - Area OpenCV: 87 - Length: 48.6274
* Contour[452] - Area (M_00) = 75.5 - Area OpenCV: 75.5 - Length: 51.2132
* Contour[462] - Area (M_00) = 68.5 - Area OpenCV: 68.5 - Length: 61.2132
* Contour[481] - Area (M_00) = 44.5 - Area OpenCV: 44.5 - Length: 37.5563
* Contour[491] - Area (M_00) = 89 - Area OpenCV: 89 - Length: 58.2843
* Contour[586] - Area (M_00) = 51 - Area OpenCV: 51 - Length: 41.799
* Contour[616] - Area (M_00) = 573 - Area OpenCV: 573 - Length: 154.569
* Contour[626] - Area (M_00) = 229.5 - Area OpenCV: 229.5 - Length: 93.3553
* Contour[628] - Area (M_00) = 3361 - Area OpenCV: 3361 - Length: 499.706
* Contour[646] - Area (M_00) = 67.5 - Area OpenCV: 67.5 - Length: 42.3848
* Contour[683] - Area (M_00) = 170.5 - Area OpenCV: 170.5 - Length: 82.5269
* Contour[709] - Area (M_00) = 44 - Area OpenCV: 44 - Length: 34.9706
* Contour[710] - Area (M_00) = 63 - Area OpenCV: 63 - Length: 44.9706
* Contour[717] - Area (M_00) = 66 - Area OpenCV: 66 - Length: 38.1421
* Contour[740] - Area (M_00) = 2207 - Area OpenCV: 2207 - Length: 619.103
* Contour[766] - Area (M_00) = 957.5 - Area OpenCV: 957.5 - Length: 345.238
* Contour[777] - Area (M_00) = 146 - Area OpenCV: 146 - Length: 75.9411
* Contour[796] - Area (M_00) = 46 - Area OpenCV: 46 - Length: 36.9706
* Contour[807] - Area (M_00) = 252.5 - Area OpenCV: 252.5 - Length: 112.184
* Contour[824] - Area (M_00) = 107 - Area OpenCV: 107 - Length: 59.4558
* Contour[863] - Area (M_00) = 1176.5 - Area OpenCV: 1176.5 - Length: 420.149
```

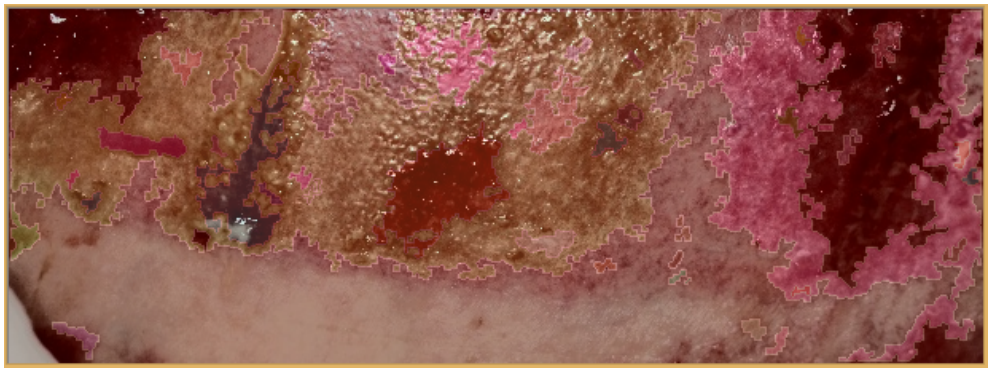


Figura 5.10: K-Means con áreas coloreadas

Se podrían descartar las áreas menores de determinado tamaño, aunque sería un problema asociar las restantes a los colores de la imagen de cara al usuario. Además, descartar pequeñas áreas no parece la mejor solución, sería mejor integrarlas.

En definitiva, `K-Means` parece que es la herramienta a usar, pero no se estaba usando de forma adecuada, por ello se decidió usar un `K-Means` puro.

5.2.1.2. K-Means puro

La función que ejecuta `K-Means` es `cv:kmeans`. Esta función requiere como parámetro la imagen a procesar, sin embargo dicha imagen debe estar en una matriz unidireccional, no la convencional de 3 dimensiones (R,G,B).

Para ello, implementamos el siguiente código:

```
Mat p = Mat::zeros(img_od.cols*img_od.rows, 5, CV_32F);
Mat bestLabels, centers, clustered;
vector<Mat> bgr;
cv::split(img_od, bgr);
for(int i=0; i<img_od.cols*img_od.rows; i++) {
p.at<float>(i,0) = (i/img_od.cols) / img_od.rows;
    p.at<float>(i,1) = (i%img_od.cols) / img_od.cols;
    p.at<float>(i,2) = bgr[0].data[i] / 255.0;
    p.at<float>(i,3) = bgr[1].data[i] / 255.0;
    p.at<float>(i,4) = bgr[2].data[i] / 255.0;
}
```

No obstante, dicho código se optimizó con funciones propias de la libería como `reshaped_image`, función que no había sido descubierta hasta entonces y que permitió acortar los tiempos de ejecución del algoritmo en buena medida:

```
cv::Mat reshaped_image = image.reshape(1, image.cols * image.rows);
cv::Mat reshaped_image32f;
reshaped_image.convertTo(reshaped_image32f, CV_32FC1, 1.0 / 255.0);
```

Una vez aplicado el `K-Means`, se obtuvieron los resultados esperados, diferenciado las zonas pintándolas del color predominante, en vez de grises o colores aleatorios como se hacía antes.

El propósito de colorear con la tonalidad predominante no es sólo visual. Antes, no se podía decir qué porcentaje había de cada, porque no se sabía si un color correspondía a la zona sana o no.

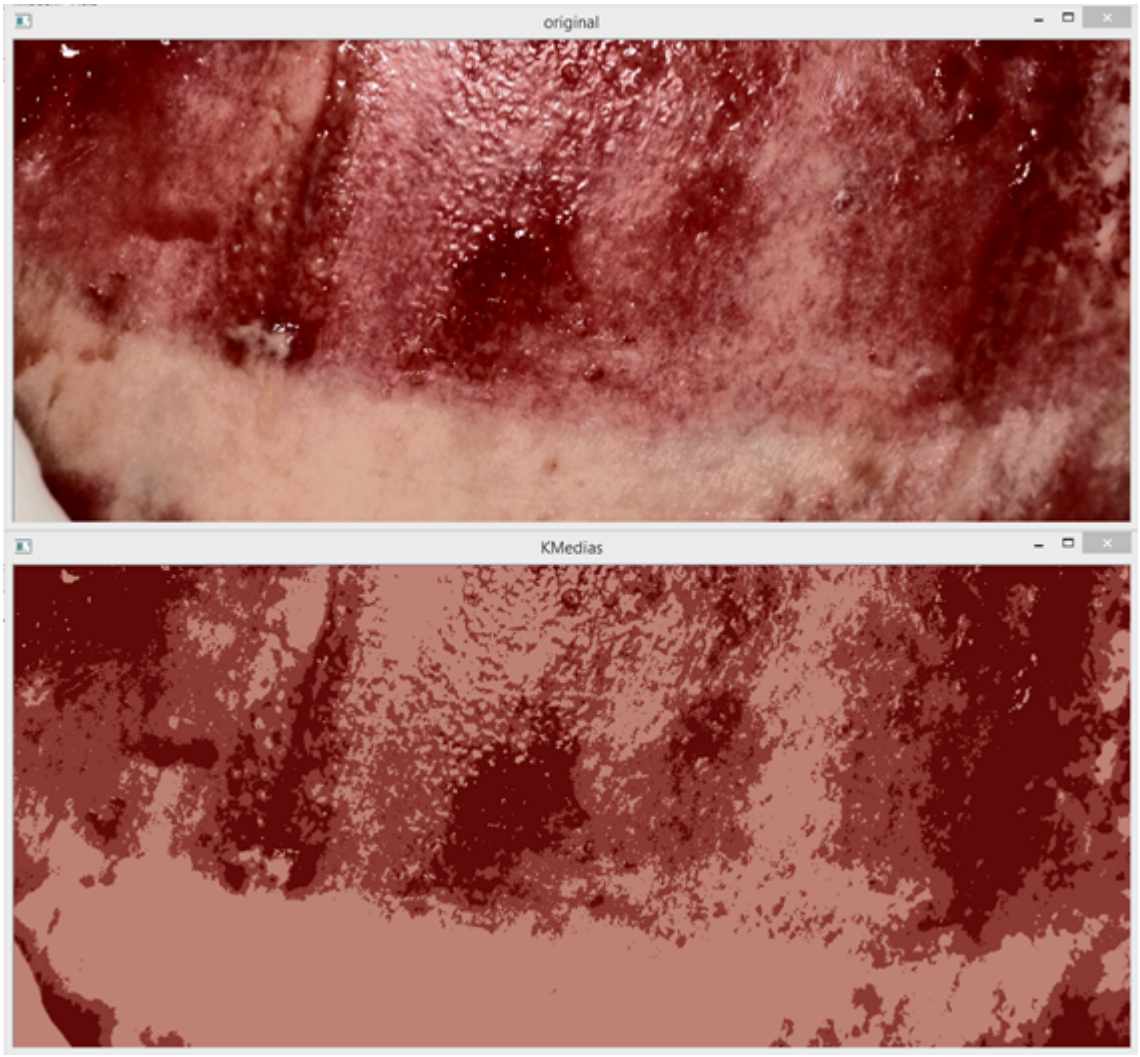


Figura 5.11: K-Means puro con k=3

Se pensó que se podría cuantificar lo sana que está una zona en función de la oscuridad del color, de forma que se asume que una zona clara es piel más sana que una zona oscura. Entonces, en la Figura 5.11 se podría obtener un resultado parecido a este:

- 25 % de tejido de granulación (color granate)
- 25 % de piel sanándose (color rojo)
- 50 % de piel sana (color rosa)

Se decidió seguir esta idea, por ello se prosiguió trabajando en el cálculo de porcentajes de las

áreas de diferente color. Dichos porcentajes se calcularon de forma sencilla teniendo en cuenta el número de píxeles, de forma que:

$$\text{Porcentaje de un color} = \frac{\text{Número de píxeles de un color}}{\text{Número de píxeles de la imagen}} * 100$$

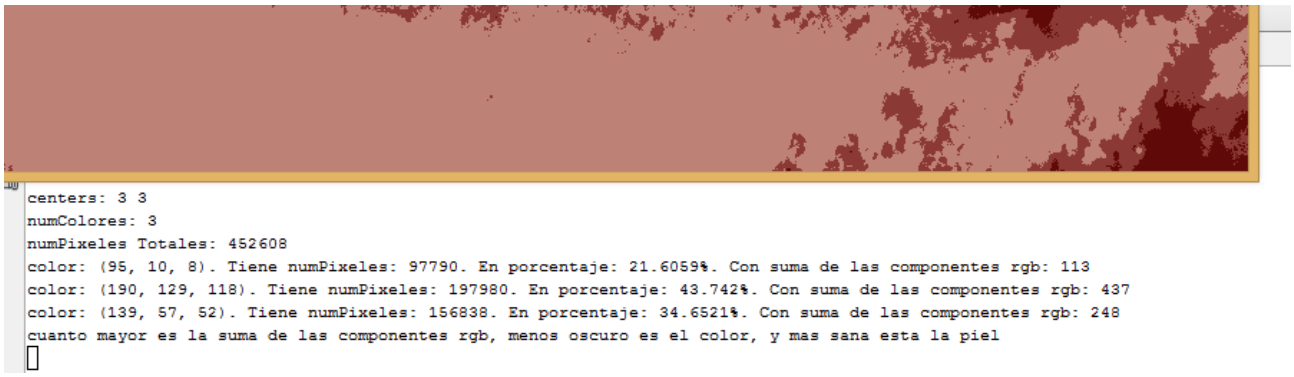


Figura 5.12: Porcentajes obtenidos con k=3

No obstante, el método no es perfecto, ni mucho menos. Tan perfecto como las propias fotografías a analizar. Se probaron con otras imágenes, obteniendo resultados como los de la Figuras 5.13 y 5.14.

Figura 5.13: Fotografía tomada con flash

Queda claro en la Figura 5.14 cómo el flash es identificado por `K-Means` como un color de la imagen y lo asocia a piel sana. Cuando parecía que se había dado con la forma correcta de aplicar el `K-Means`, apareció este problema, cuyo impacto se intentó minimizar el impacto en el resultado final.

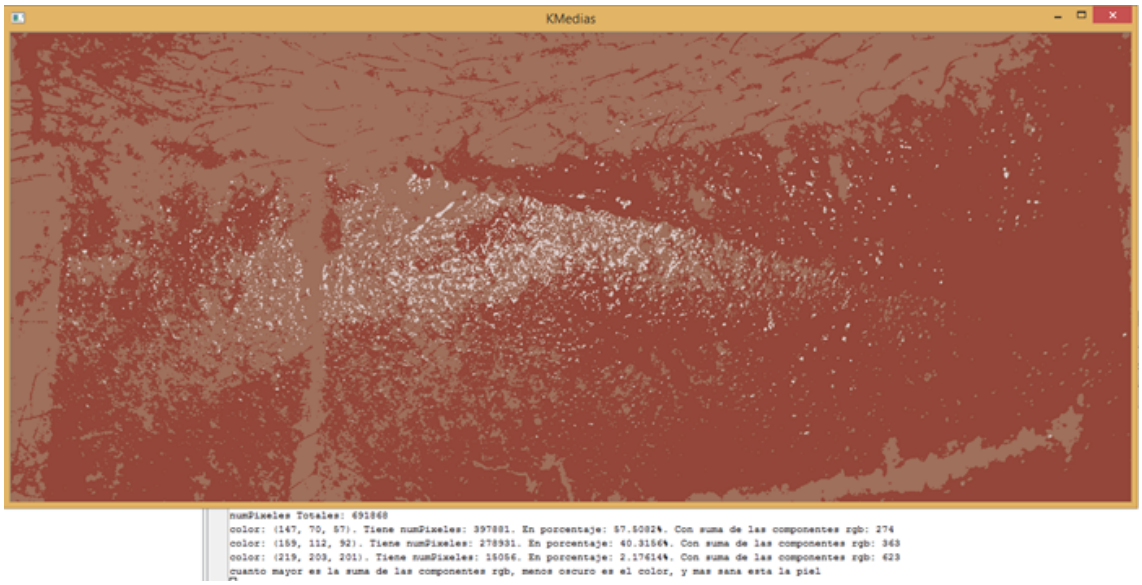


Figura 5.14: K-Means de la Figura 5.13 con k=4

5.2.1.3. Problema de brillos en la imagen

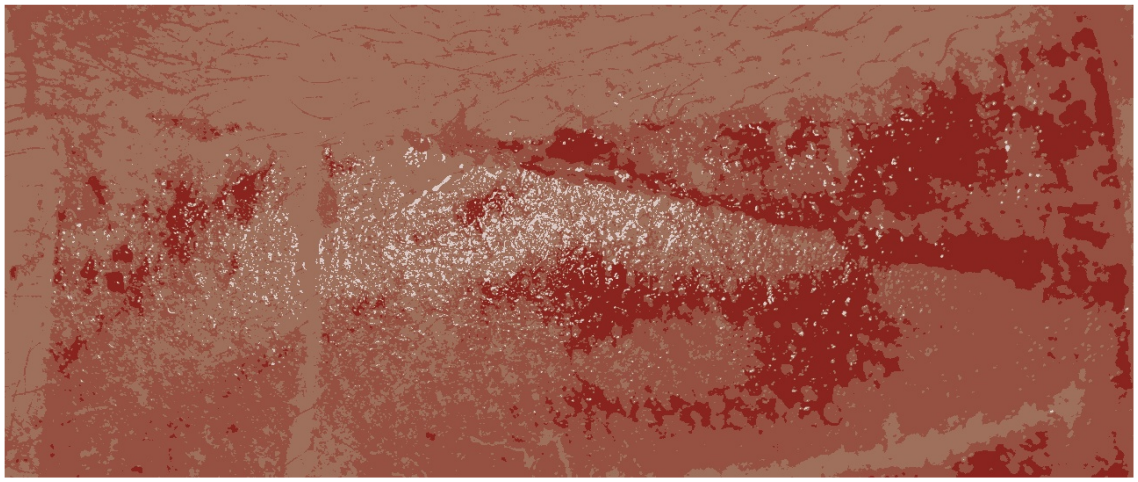
El problema es claro, las fotografías pueden haber sido tomadas con flash, o puede haber algún brillo de otro tipo que altere el procesamiento de la imagen.

La primera solución, fue ignorar la zona con flash, para ello se puede aplicar un `Threshold` para *descartar* aquellos píxeles por encima de un determinado umbral, dicho umbral sería un color próximo al blanco (255, 255, 255). Sin embargo, la solución más correcta y rápida sería la de añadir un centro más al `K-Means`, es decir, si en la Figura 5.14 se aplicó el `K-Means` con k=3, se aplicaría con k=4. El nuevo centro añadido se haría corresponder con el color menos oscuro de la imagen.

En la Figura 5.15 vemos el resultado obtenido con k=4 partiendo del original de la Figura 5.13. Se ve efectivamente que el color más claro es el color: (219, 203, 201) correspondiente al flash. Dicho color representa un 2.17614 % de la imagen. Porcentaje muy bajo que se podría ignorar para solucionar este problema.

Surgieron varios caminos según se fue investigando sobre el flash. Este tema no resulta trivial, ha sido problemático desde hace tiempo en el mundo de la fotografía y de la edición de imagen. Una primera aproximación al el problema sugirió localizar puntos que fuesen de color blanco o prácticamente blancos para localizar los puntos de flash. Esto conllevaba una consecuencia, y es que cualquier color blanco de la imagen queda identificado como brillo.

Tras una pequeña investigación al respecto, se encontraron dos algoritmos de código libre que podían satisfacer las necesidades. Estos algoritmos son el `SimpleBlobDetector`, propio de la



```
numColores: 4
numPixeles Totales: 691868
color: (150, 81, 67). Tiene numPixeles: 296471. En porcentaje: 42.8508%. Con suma de las componentes rgb: 298
color: (159, 112, 92). Tiene numPixeles: 278931. En porcentaje: 40.3156%. Con suma de las componentes rgb: 363
color: (138, 36, 30). Tiene numPixeles: 101410. En porcentaje: 14.6574%. Con suma de las componentes rgb: 204
color: (219, 203, 201). Tiene numPixeles: 15056. En porcentaje: 2.17614%. Con suma de las componentes rgb: 623
cuanto mayor es la suma de las componentes rgb, menos oscuro es el color, y mas sana esta la piel
```

Figura 5.15: K-Means de la Figura 5.13 con k=4

librería `OpenCV`; y una librería externa en `C` y `C++` llamada `cvBlob`.

Las pruebas con el primer método, `SimpleBlobDetector`, dieron como resultado una localización bastante buena de los puntos de luz, los cuales quedaban rodeados por un círculo de color. El problema surgió a raíz de que el algoritmo no devolvía ningún valor sobre el que trabajar, se limitaba a pintar sobre la imagen dichos círculos, y con esa ausencia de datos no se podían tomar ninguna clase de decisiones ni medidas.

Según la investigación fue a más, se encontraron otras pequeñas librerías, que aportaban ideas sobre como resolver este problema (obtener los brillos con sus puntos, área, etc.), pero pertenecían a otros lenguajes de scripting y resultaba tediosa la traducción, ya que los métodos entre las librerías (ej. `Matlab` y `OpenCV`) poseían denominaciones diferentes, pero esta opción quedaría descartada a menos que no se encontrase algo mejor. Una de las librerías más prometedoras fue `cvBlob`, una librería que implementaba, a partir de los métodos básicos de `SimpleBlobDetection` de `OpenCV`, un algoritmo capaz de localizar brillos en función de su área, color, forma y otras características que no resultan tan útiles para el problema actual. Además, devuelve los puntos que forman los vértices de un rectángulo que contiene el brillo en su interior, junto con la superficie del rectángulo. El algoritmo ofrece más valores cuantitativos y cualitativos de cada blob (punto de brillo), pero para el caso en cuestión no eran necesarios o carecían de tanta utilidad como los otros. Una ejecución del algoritmo, simplemente localizando los brillos es el mostrado en la figura 5.16, con el código:

```
//Inicializacion de variables y carga de la imagen.
```

```
const char * c = file.c_str();
Mat inputImg = imread(file);
IplImage *img=cvLoadImage(c, CV_LOAD_IMAGE_UNCHANGED);
IplImage *gray = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);

//Cambio de color a blanco y negro y threshold
cvCvtColor(img, gray, CV_BGR2GRAY);
cvThreshold(gray, gray, 150, 255, CV_THRESH_BINARY);
IplImage *labelImg=cvCreateImage(cvGetSize(gray), IPL_DEPTH_LABEL, 1);

//Ejecución del propio algoritmo de blobs.
CvBlobs blobs;
cvLabel(gray, labelImg, blobs);

//Dibujo de las cruces en la imagen.
cvRenderBlobs(labelImg, blobs, img, img);
```

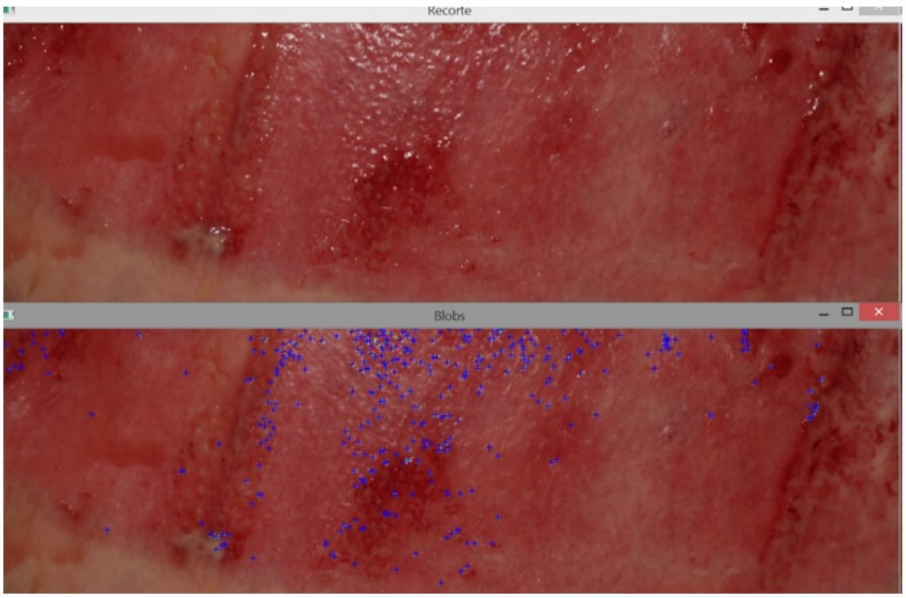


Figura 5.16: Detección de brillos. Aparecen marcados con una cruz todos los puntos de brillo

La variable `blob` que se muestra en el código contiene todos los datos necesarios para el resto del algoritmo. Una vez localizados los brillos y cuantificados, el siguiente objetivo consiste en hacer algo con ellos para que no alteren los resultados de la imagen, dado que para valores de K=3 (tres zonas de color) los brillos interfieren negativamente cambiando una de las capas de datos utilizables a brillos, o se combinan capas, o suceden variaciones de este estilo. Ofrece

además métodos para rellenar los rectángulos y editar directamente esos trozos, porque no se pueden localizar con precisión la forma de los brillos (figura 5.17).

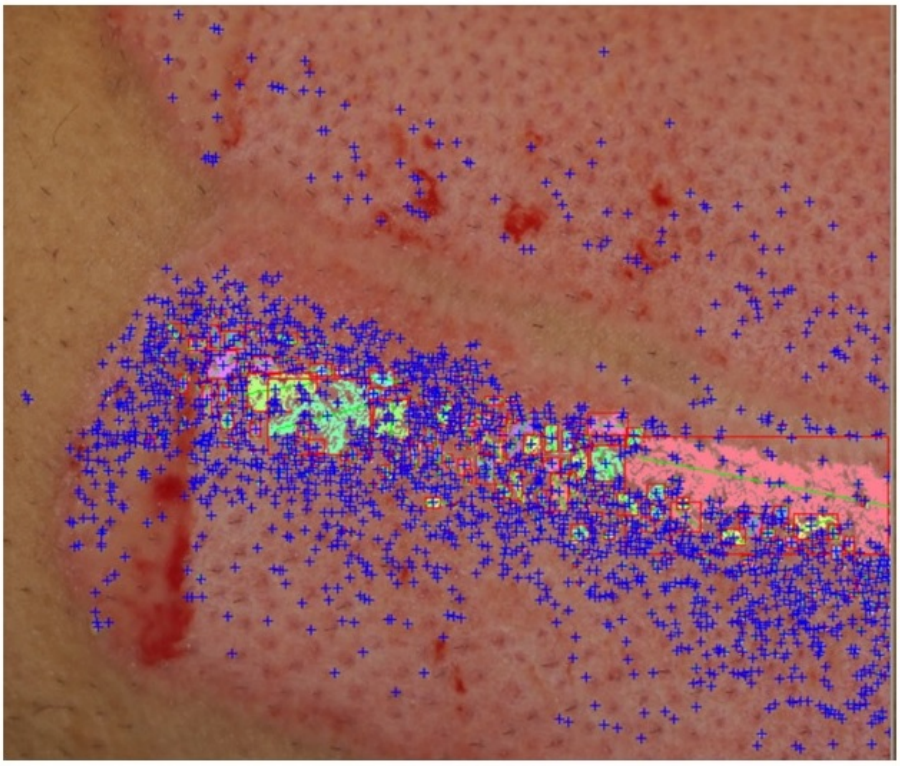


Figura 5.17: Zonas de brillos pintadas y rodeadas por un rectángulo

Esto permite rellenar los rectángulos de cualquier color para poder operar de diferentes formas. Una de las ideas que se intentó llevar a cabo fue la de cubrir esos rectángulos con datos de los píxeles que se encontraban alrededor de la zona negra.

5.2.2. Filtros de Gabor

Una de las ideas para diferenciar las zonas de las imágenes fue la distinción por texturas. Para ello se propuso el uso de `filtros lineales`.

Los `filtros lineales` procesan las señales variables en el tiempo para producir señales de salidas sujetas a restricciones de linealidad. En nuestro caso, el valor de los nuevos píxeles se compone de una suma ponderada de las intensidades de los píxeles en el área circundante de la imagen original.

Al considerarse los tipos de filtros lineales comunes como son los filtros `Laplacianos` y `Haar-like`, se escogió utilizar `filtros de Gabor` debido a que son selectivos para escala y orientación, de la misma forma que lo son las imágenes a procesar.

Las `funciones de Gabor` son funciones que operan en el conjunto de los números complejos, por lo tanto, tienen una componente real y otra componente imaginaria que representa direcciones ortogonales. Las dos componentes pueden formar un número complejo o ser utilizadas individualmente.

En las imágenes dadas se aplica la función correspondiente a la parte real del filtro [10]:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \cos(2\pi \frac{x'}{\lambda} + \psi) \tag{1}$$

donde:

- $x' = x \cos\theta + y \sin\theta$
- $y' = -x \sin\theta + y \cos\theta$
- λ representa la longitud de onda de la función sinusoidal.
- θ representa la orientación de la normal a las franjas paralelas de una `función Gabor`.
- ψ es la desviación de fase.
- σ es la varianza de la función gaussiana.
- γ es la relación de aspecto espacial, y especifica la elipticidad de la `función de Gabor`.

De esta forma, se obtiene el núcleo o kernel que se utiliza después para obtener la imagen final por medio de la convolución entre este núcleo y la imagen original.

Sabiendo el efecto que tienen los diferentes parámetros de la función sobre las imágenes se pueden usar para la distinción de las texturas y la creación del algoritmo (véase apartado 4.1.8).

Los pasos seguidos para la utilización de los `filtros de Gabor` fueron los siguientes:

- Convertir la imagen de `RGB` a formato de escala de grises utilizando `cvtColor`. Esto es necesario para el uso eficiente de los filtros.
- Se cambia el tipo de la imagen de `8UC1` a `32FC1`. Esto significa que la imagen original está en formato de 8 bits por píxel, es decir, un píxel puede tener valores de 0 a 255, el cual es el rango normal para la mayoría de formatos de imagen y vídeo; y se cambia a tipo `float`, por lo tanto, el píxel puede tener cualquier valor entre 0 y 1. Esto es útil para algunos conjuntos de cálculos sobre los datos, pero tiene que ser convertido en 8 bits para guardar la imagen multiplicando cada píxel por 255.
- Se crean las barras para la introducción de los datos de los parámetros de forma dinámica y se llama a la función que cambia la salida de las imágenes resultantes según estos parámetros. Esta función realiza tres operaciones:
  - Crea el núcleo para su convolución con la imagen original. El núcleo se realiza mediante la `función de Gabor` explicada (véase la función (1)).
  - Aplica la convolución entre la imagen y el núcleo a través de la función `filter2D`. Esta función calcula la correlación por lo que el núcleo no se refleja alrededor del punto de anclaje.
  - Muestra tres imágenes diferentes: la imagen del núcleo, la imagen resultante del `filtro de Gabor` aplicado y la última imagen con cada elemento de la matriz elevado al cuadrado.

Las pruebas realizadas arrojaron diversos resultados dependiendo de las condiciones de las fotografías, ya que son muy diferentes unas de otras.

Los valores que se utilizaron para generar el `filtro de Gabor` y de los umbrales fueron encontrados a base de prueba y error.

En los experimentos hechos, los datos introducidos que han sido más precisos a la hora de filtrar las imágenes son para:

- Varianza de la función gaussiana, valores de 3 o 4.
- Longitud de onda, valores cercanos a 100.
- Orientación, valores entre 30 y 50, aunque es dependiente del sentido de las heridas.
- Desplazamiento de fase, valores entre 65 y 80.

Figura 5.18: Imagen original

Figura 5.19: Imagen del núcleo con valores de los parámetros de la varianza de la función gaussiana 3, longitud de onda 80, orientación 37 y desplazamiento de fase 70

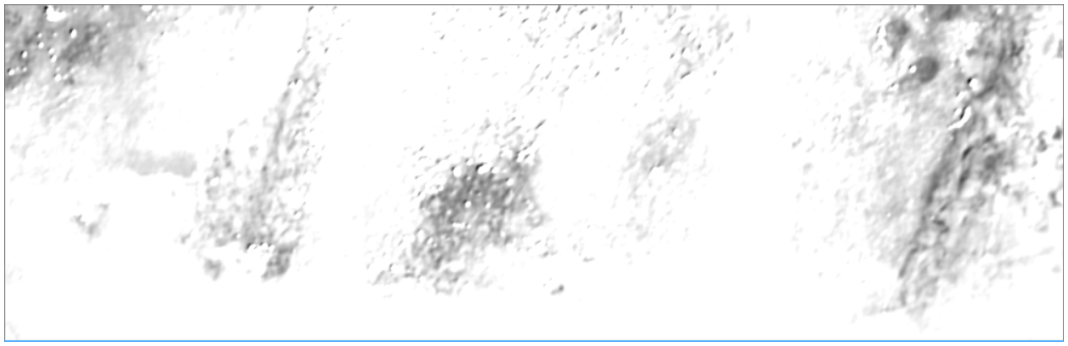


Figura 5.20: Imagen con filtro de Gabor aplicado con valores de los parámetros de la varianza de la función gaussiana 3, longitud de onda 80, orientación 37 y desplazamiento de fase 70

En la imagen de la Figura 5.20 se distinguen las zonas de las heridas que tienen una textura diferente que las zonas adyacentes. Aunque no es una zona totalmente exacta, se distinguen de forma real unas regiones de otras, pudiéndose ver de forma más precisa en la Figura 5.21.

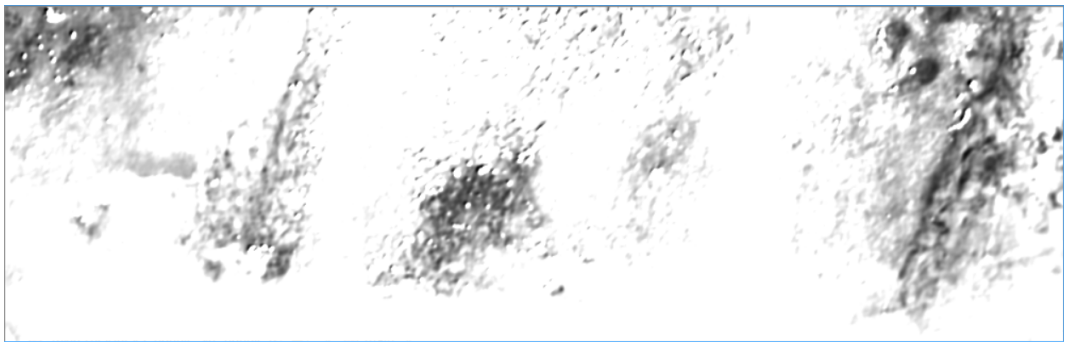


Figura 5.21: Imagen de la Figura 5.20 elevada a potencia de dos

Cambiando los valores, si se mira detenidamente, se puede ver cómo se localizan otros niveles de texturas (Figura 5.22), siendo posible diferenciar entre un mayor número de áreas.

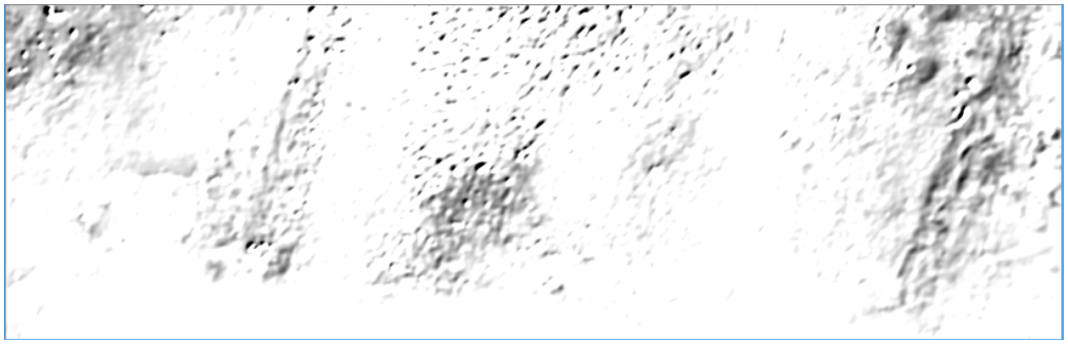


Figura 5.22: Imagen con filtro de Gabor aplicado con valores de los parámetros de la varianza de la función gaussiana 4, longitud de onda 69, orientación 45 y desplazamiento de fase 75

Sin embargo, en imágenes con brillo, la luz es tomada como si fuera una textura a diferenciar. Esto hace que en este tipo de imágenes no sea fiable el algoritmo. Este obstáculo apareció ya mediante el desarrollo del procedimiento por K-Means (se puede ver su planteamiento en el apartado 5.2.1.3).

Otro de los problemas de esta técnica es que no es útil en imágenes en las que la heridas son recientes, ya que en su mayoría carecen de relieve y no hay posibles texturas a distinguir.

5.3. Aportaciones de los miembros del grupo

5.3.1. Roberto de Miguel López

En primer lugar, ya tuvimos en el curso anterior una asignatura optativa llamada *Percepción computacional* con la cual me bauticé en el mundo del procesamiento de imágenes. En dicha asignatura aprendí los conceptos teóricos más básicos como son: la representación de las imágenes y sus distintos modelos de color, suavizado de la imagen a partir de un filtrado gaussiano, el histograma de las imágenes y algunas de las operaciones que podíamos realizar con éste como la ecualización y expansión.

Además, este mismo curso se impartió la asignatura obligatoria *Ingeniería del conocimiento*. Esta asignatura que a priori no tenía demasiado que ver con la visión computacional fue extremadamente útil, ya que se estudiaron diferentes algoritmos: para encontrar el camino más corto como el "A estrella", aprendizaje en árboles de decisión con el *ID3*... y entre otros el algoritmo de agrupamiento o clasificación K-Medias, Bayes y Lloyd. Digo que esta asignatura fue de gran ayuda, porque antes de empezar el proyecto, ya conocía a la perfección la teoría del K-Means, algoritmo que yo mismo había tenido que implementar en Java.

Fase de preparación y aprendizaje

En esta fase instalé la librería OpenCV en mi ordenador, no sin librarme de alguna pequeña complicación técnica. Una vez instalado y hechas las primeras pruebas de contacto procedimos a los primeros problemas planteados.

En el primer ejercicio de *encontrar las diferencias* no conseguí hallar la solución, ya que me encontraba todavía estudiando los conceptos teóricos y realizando las primeras pruebas de ensayo. En la solución surgieron problemas al implementar algunas operaciones que nos producían valores para los píxeles entre 0 y 1, no los habituales entre 0 y 255. Dicho problema hizo que repasara las representaciones de color, y aportara distintos ejemplos de una imagen en las diversas representaciones de color (RGB, HLS, HVS...).

En la *busqueda de Wally* participé activamente aportando una solución individual. Tras la resolución de este ejercicio se dio por finalizada esta fase de prueba de contacto.

Fase de desarrollo

Centrados ya en el proyecto en cuestión, mi compañero Daniel había conseguido sacar los contornos de la piel sana y de la piel no sana. Mi compañero Rubén y yo nos centramos en el preprocesamiento y ecualización de la imagen. Yo me dediqué a la obtención del color de un píxel para la identificación de los colores de la tarjeta QP, así como la separación de la

tarjeta, del resto de la herida. Desarrollé la primera solución de recorte horizontal por medio de la interacción del usuario. De esta forma, aplicamos el primer `K-Means` de mi compañero Daniel tras el recorte de la imagen.

A partir de ese momento, tras alguna reunión con el director de proyecto, decidimos que el `K-Means` parece el camino a seguir. Dejé la ecualización en manos de mi compañero Rubén, y me dediqué al completo a la mejora del `K-Means` en colaboración con Daniel.

En primer lugar, debíamos encontrar las áreas ya delimitadas por contornos y colorearlas. Daniel consiguió pintar las diferentes zonas con tonos de la escala de grises, sin embargo, esto no era suficiente, ya que no podíamos decir a que estado de la piel (sana o no) correspondía cada área. Con afán de colorear las áreas de forma significativa, conseguí calcular cuantitativamente las áreas encontradas y pintarlas de colores predefinidos tal como he explicado en el apartado 5.2.1.1, no obstante, no era la solución definitiva.

Después de la reunión con el equipo médico, Daniel y yo llegamos a la conclusión de que debíamos estar aplicando el `K-Means` de forma errónea, pues obteníamos bastantes más áreas diferenciadas que los K núcleos definidos para el algoritmo, algo que entendiendo el funcionamiento de éste, era imposible.

Conseguí optimizar partes del código, y dar con la implementación del K-Means adecuada tal como he descrito en el apartado 5.2.1.2 obteniendo los resultados esperados que se puede ver en la Figura 5.11 dentro de ese mismo apartado.

Una vez llegado a este punto, debíamos llegar a obtener una información cuantitativa. Por ello, propuse la clasificación por color de la imagen, identificando las áreas coloreadas de un color más claro, como piel más sana. Además de la implementación del método para calcular el porcentaje de un color en la imagen.

Esta solución se llevó a estudio, y preparé un documento en el que aplicaba el algoritmo a todas las fotografías de las que disponíamos. Tras comentar este documento con el tutor del proyecto, se detectó el conocido problema de los brillos. Dicho problema propuse resolverlo fácilmente añadiendo un núcleo más al `K-Means` para que se asociase con este brillo, y entonces descartarlo.

Aquí terminó mi contribución al desarrollo de la solución final, y empezó mi trabajo en la redacción de la memoria.

Redacción de la memoria

En la escritura de esta memoria, he sido autor de todas las partes que menciono anteriormente, así como de una parte del apartado 6 *Discusión sobre los resultados*, que distribuimos de forma equitativa entre los componentes del equipo. Además, redacté la introducción, y casi en su totalidad el apartado 4.1 *Conocimientos teóricos necesarios*, previo estudio y documentación.

Asimismo, comentar que la composición de esta memoria, y en concreto de este apartado, ha sido más fácil gracias a un diario en el que recogí todo mi trabajo, así como un resumen de las reuniones con el tutor y el equipo médico.

5.3.2. Rubén Moreira López

Fase de preparación y aprendizaje

Durante esta fase instalé la librería de `OpenCV` sin problemas pero la elección del entorno de desarrollo para poder crear los algoritmos fue muy variada, pasando desde `Visual Studio`, por `Qt Creator (Ubuntu)`, hasta llegar a la decisión definitiva que fue `Netbeans`. Gracias a esto pude empezar con los primeros problemas de aprendizaje.

En el primer problema, *encontrar las diferencias*, desarrollé una solución que consistía en el cambio de color de las imágenes a escala de grises y la aplicación posterior de diferentes `Threshold`. Para aplicar los valores a los cinco tipos de `Threshold` se crearon dos barras de desplazamiento, una para elegir el tipo y otra para el valor.

Esta solución con barras de desplazamiento se utilizaría de manera semejante después en la detección por texturas de la imagen.

La *búsqueda de Wally*, por mi parte, no fue satisfactoria, pues intenté utilizar un método diferente al `Template Matching` propuesto, siendo este la segmentación de la imagen por medio del algoritmo de `Watershed`. Por desgracia, el resultado de este procedimiento fue erróneo.

Fase de desarrollo

En las partes iniciales de esta fase, me centré en el preprocesamiento de las imágenes para la creación de algoritmos por parte de mis compañeros, concretamente en la ecualización.

En la búsqueda de una solución, se encontró la ecualización por histogramas. Sin embargo, se utiliza generalmente en imágenes en escala de grises, hecho que no nos interesaba. Para resolver este problema, comprobé si cambiando el formato de color y separando en canales cada componente, haciendo que un único canal fuese en el que se aplicase la ecualización (como se explica en los pasos de 5.1), daba efectos positivos. Al ser satisfactorio, probé varios tipos de formatos de color, como `HSV`, siendo el mejor el primer elegido, `YCbCr`.

Aunque la primera impresión de la imagen ecualizada fue de un funcionamiento correcto ya que se visualizaba mejor las distintas zonas de las heridas, en posteriores reuniones se descartó esta idea dado que mostraba colores irreales y dificultaba el procesamiento.

También para la ecualización probé con la utilización de una ecualización por histogramas

adaptivo(AHE), pero el proceso impedía su uso en imágenes de color y no pudo ser solucionado.

Durante el preprocesamiento de las imágenes, también participé en el recortado de éstas, intentando usar un algoritmo automático de detección de la herida por segmentación como, por ejemplo, el algoritmo GrabCut, basado en cortes de grafos. Este algoritmo tenía un tiempo de carga muy grande y la zona delimitada no se acercaba a la realidad.

Ninguno de los demás algoritmos fue provechoso y la solución al problema del recorte fue facilitada por mi compañero Roberto, eliminando la parte superior de las imágenes. Sin embargo, con la introducción de nuevas imágenes en fases posteriores del desarrollo, el algoritmo quedó obsoleto y me encargué de crear uno nuevo basado en la extracción en forma rectangular mediante selección de las esquinas superior izquierda e inferior derecha (5.1).

Para la detección de las diferentes zonas de las heridas, el director del proyecto propuso el tratamiento de las distintas texturas mediante filtros lineales, por lo que me separé del resto de mi grupo para seguir ese camino de desarrollo. Investigando sobre los filtros lineales llegué a la conclusión que lo más factible sería el uso de `filtros de Gabor` ya que habían sido útiles en otras aplicaciones para la detección de texturas [6].

Para la creación del algoritmo se usaron las `funciones de Gabor` explicadas ((1)) y se añadieron las barras de desplazamiento vistas en la fase de preparación para poder cambiar los valores de manera dinámica.

Aunque en un principio los datos mostrados por los `filtros de Gabor` tenían efectos positivos, en una reunión se decidió que era mejor descartarlos de la solución final pues las imágenes tratadas con `K-Means` reflejaban la información de forma más precisa.

Tras la finalización del desarrollo por `K-Means` por parte de mis compañeros, me dispuse a unir los dos conceptos teniendo como base los `filtros de Gabor`. Quitando problemas con el formato de las imágenes para poder mostrarlas, no tuve inconvenientes en la unión de los métodos.

Redacción de la memoria

En la realización de esta memoria me he ocupado de aquellas partes que he desarrollador personalmente, tales como el preprocesamiento de las imágenes, conceptos sobre la ecualización por histogramas y `filtros de Gabor`, la fase de desarrollo sobre la detección de texturas y partes sobre los resultados y consideraciones finales.

5.3.3. Daniel Novillo Villarejo

Pruebas iniciales.

En los algoritmos iniciales de aprendizaje, el problema de encontrar las diferencias fue fuertemente desarrollado por mí, desde la idea de la resta absoluta combinado con `canny` hasta la combinación de las imágenes para que fuese más visible. Entretanto se decidió aplicar una técnica de limpieza de ruido, el `Blur`, para eliminar pequeñas diferencias que serían detectadas por la resta absoluta y por lo tanto por el `canny`, pudiendo encontrar pequeños puntos por toda la imagen (dependiendo de como fuese tomada) que no corresponden a las diferencias reales. Estas técnicas habían sido utilizadas hacía poco tiempo por lo que las primeras pruebas prototípicas se efectuaron rápidamente (Figuras 4.18 y 4.19).

Preprocesamiento de imágenes.

Al principio, quisimos empezar el problema limpiando los colores de la imagen para que fuese más fácil su tratamiento. Esto produjo complicaciones dado que muchos algoritmos interesantes que se encargaban de esto eran de pago o no se encontró el código y dado que hacerlos por nuestra cuenta sería tedioso, investigué sobre otros caminos, pero todos llevaban a que la calibración más fácil era aquella que se hacía sobre la propia cámara, algo que en este caso no era viable, por lo que se estudiaron otras técnicas de calibración, concretamente el histograma, por parte de mis compañeros.

Primer K-Means.

El mérito de encontrar el `K-Means` y empezar la investigación sobre el mismo recae sobre mi persona, ya que en experiencias previas había usado este algoritmo para temas de color. Tras tratar de encontrar formas de afrontar el problema, se me ocurrió la idea de, como comienzo, tratar de localizar las áreas de sangre. Aquí entra en juego el `K-Means`, para tratar de buscar y diferenciar esas áreas.

Haciendo pruebas aleatorias, llegué al resultado de la figura 5.6, que hizo que el `K-Means` fuese considerado como método importante en la prueba de concepto que estábamos llevando a cabo. Un poco de ajuste del algoritmo dejaba ver como se podían sacar diferentes áreas en función de la cantidad de núcleos que se le especificaban al algoritmo, pero la poca precisión de esto (dado que el `K-Means` que nosotros utilizamos tiene una pseudo-aleatoriedad en la inicialización) conlleva que este camino de investigación quedase descartado, ya que además esto sólo servía para rodear todas aquellas áreas que no fuesen de un tipo determinado (la ejecución normal hace que se diferencie la piel en proceso de sanación de lo que no lo es, por lo que se rodearán todas aquellas partes que sean diferentes, figura 5.7).

K-Means puro.

Por otra parte, entre Roberto de Miguel y yo, tras darnos cuenta de que algo tenía que ir mal con el algoritmo (pese a que algunos de los resultados nos eran muy útiles), hicimos los ajustes

necesarios para que el algoritmo sacase los datos que debería sacar y no la imagen binaria que se obtenía previamente. Finalmente conseguimos que el algoritmo funcionase como se esperaba.

Brillos.

Fue entonces cuando se descubrió un nuevo problema: los brillos afectaban negativamente al `K-Means`, ya que si había demasiadas zonas de brillo en alguna imagen formaban una capa separada del resto.

Este era otro problema que había traído de cabeza en muchas ocasiones a investigadores previos a nosotros, ya que el principal inconveniente es que no se sabe a ciencia cierta qué hay debajo de los brillos, como mucho te puedes hacer una idea en función de lo que hay alrededor. Con esto en mente, empecé la construcción de un algoritmo que buscaba cada punto y trataba de reconstruirlo. Para esto, primero se localizaban los brillos con ayuda de la librería `cvBlob` (previamente se usaba el `SimpleBlobDetector`, pero dado que traía muchos problemas a la hora de localizar los puntos porque no devolvía datos, solo una imagen con los puntos rodeados, quedó descartado) y luego se procedió a crear un algoritmo que usase las medias de los píxeles vecinos al brillo para reconstruirlo. Esto nunca salió del plano teórico, porque cuando se trataba de brillos muy grandes la solución podría enrarecerse y dejar la muestra poco útil, así que se acordó lo siguiente: En caso de que el porcentaje de brillo fuese muy alto, la muestra quedaría inutilizada y no se usaría para el algoritmo, mientras que si era muy bajo, se pondrían de color verde los rectángulos que cubren el brillo y se aumentaría el número de núcleos en uno, para que todo el color verde quedase fuera de las otras áreas. Utilizamos el color verde porque es un color que es prácticamente imposible de ver en una muestra de estas características.

Seguimiento del proyecto.

Todos los avances, ideas, pensamientos, trozos de código, imágenes, etc., fueron recogidos en un diario de forma cuidadosa y bien fechada.

Contribución a la memoria.

Además del apartado actual, he escrito en la memoria todas aquellas partes en las que he participado, descritas con anterioridad en este mismo apartado.

6. Discusión sobre los resultados

6.1. Resultados

6.1.1. OpenCV

`OpenCV` es una librería muy potente con mucha funcionalidad y que se puede usar en proyectos que impliquen ideas innovadoras como ésta. `OpenCV` es multiplataforma, existiendo versiones para `GNU/Linux`, `Mac OS X` y `Windows`. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, estereovisión y visión robótica. El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código `C` y `C++` optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo [1].

Dada la gran documentación y todo el respaldo que posee por parte de la comunidad y de las `APIs` oficiales, lo hace idóneo para proyectos como éste o para futuros desarrollos a partir de las técnicas expuestas en este trabajo [2].

6.1.2. K-Means como prueba de concepto

El objetivo del proyecto se planteó como una prueba de concepto, en la que se iba intentar obtener la mayor información posible, así como conocer la capacidad del procesamiento de imágenes aplicados a diferentes campos de la medicina. Se muestran en las Figuras 6.1, 6.3, 6.5 más ejemplos de la solución obtenida.

El método adoptado `K-Means`, se ha visto que es muy potente, y que podría utilizarse para identificar zonas de otro tipo de heridas, es decir, se podría extender el algoritmo a otros campos de la medicina en los que podría trabajar de igual manera.

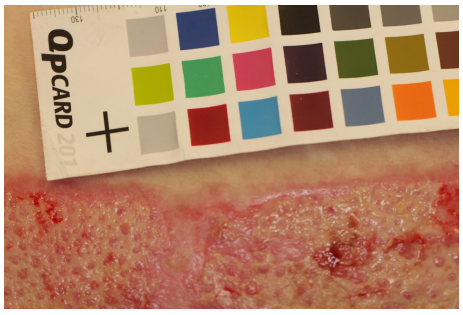


Figura 6.1: Ejemplo 1: Imagen original

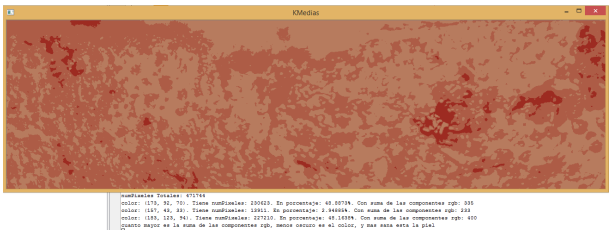


Figura 6.2: Ejemplo 1

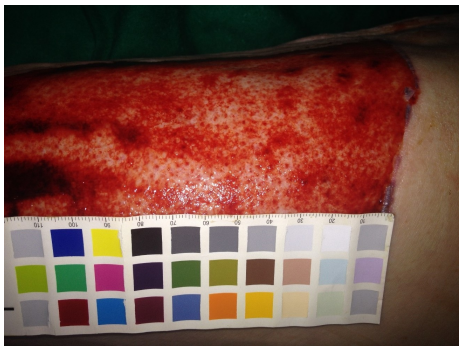


Figura 6.3: Ejemplo 2: Imagen original

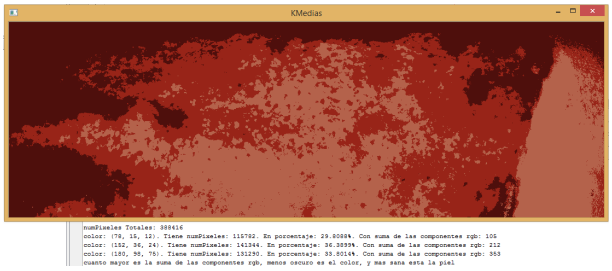


Figura 6.4: Ejemplo 2

Figura 6.5: Ejemplo 3: Imagen original

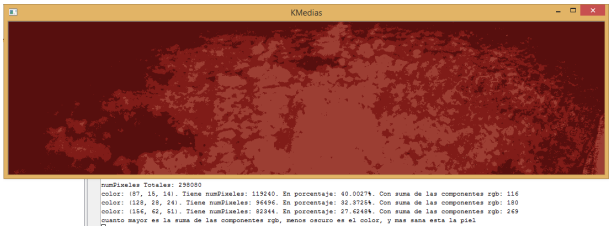


Figura 6.6: Ejemplo 3

6.1.3. Aplicación de detección de texturas a otros ámbitos

La técnica seguida para la detección de texturas, que no fue absolutamente positiva para la solución planteada, se observó que podría ser viable en la identificación en otras investigaciones. En este trabajo se ha mostrado la factibilidad del reconocimiento de texturas por medio del `filtro de Gabor` que pueden ser útiles en problemas de segmentación.

6.2. Consideraciones

A lo largo del proyecto han surgido una serie de problemas que se recogen en este apartado.

6.2.1. Calidad de las muestras

La cantidad de muestras recogidas en las primeras fases del desarrollo eran escasas, aunque suficientes para poder hacer pruebas. La mayoría de las pruebas se hicieron con unas pocas imágenes, ya que eran las más informativas, pues tenían el mayor número de zonas a diferenciar.

Durante el desarrollo, se notó que la calidad de las imágenes es un factor fundamental a la hora de su tratamiento. Ciertas heridas, aún siendo la calidad alta, no tenían un nivel aceptable, principalmente por causa de la iluminación, exceso de brillos y/o enfoque.

En una parte más desarrollada de la investigación se recibieron una cantidad de muestras, sin embargo, algunas imágenes también carecían de la iluminación adecuada.

Un desarrollo con un número de fotografías amplio y sin problemas de calidad hubiera facilitado y agilizado el proceso, pues toda la base del trabajo depende de ellas.

6.2.2. Ecualización y tarjeta QP

Para la correcta ecualización de la imagen se incluye en todas ellas la tarjeta QP. Esta tarjeta sirve para el control de las propiedades principales de la reproducción del color, el carácter y la temperatura de éste. El problema viene dado a la hora de poder emplearlas ya que no tenemos la aplicación necesaria para hacer dicho uso.

Para poder hacer una ecualización propia de la imagen se ideó una función para encontrar los valores `RGB` de los colores. Dicha función produjo resultados negativos ya que en cada esquina de la tarjeta, cuyos valores de las tres componentes deberían ser exactamente iguales, daban valores con una gran diferencia en cada una de ellas. Este hecho causó que tampoco se pudiese hacer un algoritmo para la ecualización de la imagen con la tarjeta pues sería una tarea bastante ardua de efectuar.

De esta forma, se planteó la ecualización de la imagen por otros medios eliminando la tarjeta QP de las imágenes originales para hacer una ecualización general de ellas. En la elaboración de este proceso los colores aparecían distorsionados y no se acercaban a la realidad haciendo que fuera inútil dicho procedimiento.

Con una precisa ecualización sería posible averiguar zonas en las heridas de forma más óptima y concluiría en mejores resultados.

6.2.3. Brillos, solución adoptada

La cuestión de los brillos lleva siendo discutida desde hace tiempo por expertos en tratamiento de imágenes y no constituye un problema trivial. Como se ha explicado, logramos un algoritmo que era capaz de detectar los brillos y meterlos dentro de un rectángulo, de tal forma que se podía acceder a las posiciones y tamaños aproximados de los puntos de flash.

Tras diversas pruebas, se llegó a la conclusión de que eliminar los brillos de forma eficiente no era una solución, porque no se podía obtener con precisión el valor de los píxeles originales que debería haber en esa zona de la foto, por lo que se llegó a una solución que esquivaba esto:

Cuando en una imagen se detectan zonas de brillos, se abren dos posibles caminos:

- En caso de que los brillos fuesen menor de cierto porcentaje (un porcentaje relativamente pequeño, para poder usar la muestra) se procedería a rellenar esos rectángulos de un color poco probable en una herida humana de esas características (el color verde), de tal forma que si hay brillos, estos serían cubiertos por un rectángulo verde y se obviarían en la ejecución del `K-Means`, simplemente aumentando el numero de núcleos en 1 para que todos los polígonos verdes queden en una capa separados del resto de la información.
- En el caso contrario, cuando el brillo superase ese porcentaje, la imagen quedará descartada debido a la mala toma de la misma, ya que hay demasiada herida oculta por brillo como para poder sacar información útil.

En principio, para asegurarse que este mecanismo funcione bien y a falta de algoritmos que lo automaticen (puede que el algoritmo de detección de brillos detecte objetos que parecen brillos pero que no lo son), se pedirá al usuario que especifique si existen o no existen brillos en la imagen, tras lo cual se llevará a cabo la medición del porcentaje de éstos y se tomará la decisión adecuada.

6.2.4. Filtros de Gabor

El método de detección de texturas dio grandes resultados en un gran número de imágenes. Pese a esto este método no era totalmente adecuado por diversas razones:

- Las texturas encontradas en la ejecución del algoritmo no correspondían todas ellas a las zonas de *piel no sana*. Por ejemplo, en muchas de las fotos se encontraron texturas en zonas capilares que no se corresponden con zonas de hematomas o de tejido de granulación. Este hecho podría ser solucionado con una extracción de la herida en su totalidad.
- El problema de los brillos de las fotografías también afectó al método de texturas ya que el exceso de iluminación también era caracterizado como si fuera una textura más. La solución a este problema se adoptó en el desarrollo por `K-Means` (Apartado: 5.2.1.3).
- Al ser cada imagen diferente, los valores introducidos para la detección de texturas varían en cada una de las imágenes. Esto hace que no se puedan especificar datos por defecto para todas las imágenes provocando que la introducción de tales valores sea modificada por el usuario, y por lo tanto, ésta no sea ni exacta ni fiable.

Todos estos problemas ocasionan que el uso de este procedimiento no sea del todo aplicable al problema inicial. Sin embargo, hay una posibilidad de ampliación de esta técnica para poder ser utilizada con un porcentaje mayor de fiabilidad.

6.2.5. Pruebas con el K-Means.

Pese a que la primera forma de usar el algoritmo `K-Means` daba resultados interesantes, su uso incorrecto conllevó que se tuviese que invertir tiempo en tratar de ajustar la salida para obtener más información. Pero a la vista de los resultados, se decidió proseguir con la investigación de este algoritmo.

En lugar de buscar una forma diferente de ejecución para obtener la salida legítima, se hicieron pruebas a partir de la salida que se obtenía, lo que hizo ver que el `K-Means` podía arrojar buenos resultados y estos se podían medir. El problema que este error traía era la ausencia de capas, que es para lo que el algoritmo está hecho, para diferenciar diferentes capas en función de un dato (en este caso, el color), por lo que resultaba extraño que variando el número de núcleos el algoritmo variase de una forma que parecía aleatoria, aunque curiosamente siempre que se le daba un valor la información de la foto para ese valor era la misma (por ejemplo, para K con valor de 6 la imagen mostraba las zonas de piel en proceso de curación, para K por valor de 10, otro tipo de información y así sucesivamente).

Llegado a cierto punto, lo único que se tenían era las dos zonas diferenciadas y sus bordes gracias al algoritmo de `Canny`, por lo que se empezaron a buscar otros medios para subsanar la carencia de capas que reportaba el `K-Means`. Obviamente se llegó a la conclusión de que el uso del algoritmo no era el correcto y tras una serie de ajustes se consiguió una ejecución buena, con los datos que estábamos buscando desde el principio.

Todo esto reporta, aun así, beneficios a la investigación, puesto que el `K-Means` es capaz de sacar las diferentes áreas de la herida, pero además se ha encontrado otro uso de los datos del `K-Means` para diferenciar lo bueno de lo malo de la herida, por lo que, pese a resultar en una pequeña pérdida de tiempo, los resultados no tienen por qué ser ignorados.

6.2.6. Problemas con la solución obtenida con K-Means

Aun siendo la mejor solución encontrada, tiene algunos inconvenientes que la impiden ser perfecta.

6.2.6.1. No siempre la zona oscura es la piel no sana

En el apartado 5.2.1.2 se pensó que se podría clasificar lo sana que está una zona en función de la oscuridad del color, de forma que se asume que una zona clara es piel más sana que una zona oscura. Y se ponía la Figura 5.11 como ejemplo.

Sin embargo, según se ha ido probando el algoritmo con las muestras que llegaron posteriormente, este principio que se había asumido no siempre es cierto.

En la Figura 6.7 no solo existe el problema del flash (ya comentado en el apartado 5.2.1.3 y 6.2.3) que hace que el brillo sea identificado por `K-Means` como un color de la imagen y lo asocie a piel sana, sino que además la piel sana es más oscura que la zona de la herida.

Aplicando el razonamiento elegido, se estaría cometiendo un error mayúsculo, identificando totalmente al revés las zonas sanas.

Figura 6.7: Fotografía tomada con flash

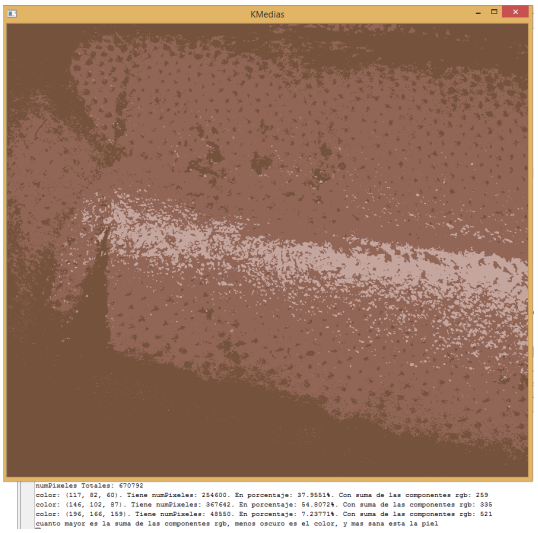


Figura 6.8: K-Means de la Figura 6.7 con k=3

6.2.6.2. Definir el número de núcleos, ¿cuántas zonas a diferenciar?

Se podría decir que generalmente hay tres zonas a distinguir: piel sana, zona de epitelización y tejido de granulación.

Sin embargo, en las nuevas muestras aparecen nuevas zonas que podrían ser de la relevancia suficiente para identificarlas, estas son: restos de apósito hidrocoloide y hematomas.

No todas las imágenes tienen las mismas zonas a diferenciar, y no se puede generalizar el número de núcleos para el `K-Means`.

Como solución, se podría preguntar al usuario este número de zonas.

6.2.6.3. Recortado imperfecto de la herida

El recortado de la imagen es muy importante, pues aislar la herida correctamente puede marcar la diferencia entre un procesado correcto y otro que no lo es.

Si en la fotografía a procesar aparecen elementos externos por un mal recortado de la foto

(sábana del hospital, ropa del paciente...), este procesamiento se verá irremediablemente alterado, ya que se identificarán estas zonas en el `K-Means` como parte de la herida, y alterarán el resto de zonas.

Con un gran número de muestras se podrían buscar patrones para eliminar determinados colores, por ejemplo, eliminar el verde característico de las sábanas de los hospitales, y no incluirlo en el procesamiento.

6.3. Trabajo futuro

Este proyecto ha arrojado alguna luz sobre nuevos caminos y disciplinas que se podrían aprovechar de la visión computacional para mejorar sus técnicas o agilizar trabajos.

6.3.1. OpenCV y la medicina

`OpenCV` ha demostrado ser una tecnología compatible con aplicaciones médicas. Mediante un equipo más profesional y una mayor cantidad de tiempo, los descubrimientos aumentarían de forma exponencial. Es un campo nuevo del que aún quedan muchas cosas por descubrir, pero la idea de las pruebas que se han realizado en este proyecto es el de agilizar (nunca sustituir, puesto que la opinión experta de un médico pesa mucho más que la aportación de un peququeño software) ciertos trabajos tediosos o ayudar en caso de emergencia. Por ejemplo, si un asistente de enfermería, en lugar de analizar una por una las heridas y revisar expedientes, con una aplicación pudiese evaluar las tomas pasadas y el estado actual, sería cómodo y rápido. Mientras que por otro lado, podría ayudar mucho a salvar vidas en desastres humanitarios en los que cada segundo cuenta, dejando que sea el software el que haga un primer reconocimiento de gravedad para así agilizar todo el proceso.

6.3.2. Machine Learning

Las limitaciones de tiempo y material impidieron seguir caminos de investigación que se abrían durante el desarrollo del proyecto.

La continuación del experimento óptima que se describió fue la de aplicar técnicas de aprendizaje automático (`Machine Learning`) para conocer, mediante un algoritmo preprogramado, las características de la propia imagen una vez separado en capas. Además, el reconocimiento de texturas podría haber sido mucho más potente mediante estos procedimientos, pero para ellos habrían hecho falta cientos de fotografías con una determinada característica, que sería la característica a reconocer, o el tiempo necesario para partir las fotos en pequeños trozos para tener muchas pequeñas muestras de piel sana, granulocitos, hematomas, etc; lo cual se empezó a

hacer, pero llevaba mucho tiempo y la ausencia de software que lo automatizase hizo que este camino se abandonase pese a la investigación previa (puesto que OpenCV tiene métodos para entrenar sistemas de aprendizaje enfocados a reconocer una característica, como es por ejemplo la detección de caras o manos, pero cuenta con cientos de miles de muestras para el entrenamiento, mientras que el proyecto no contaba con tantas).

7. Discussion of results

7.1. Results

7.1.1. OpenCV

OpenCV is a powerful library with much functionality and which is able to be used in projects involving innovative ideas as this one. OpenCV is multiplataform, with versions for GNU/Linux, Mac OS X and Windows. It contains over 500 functions covering a wide range of areas, such as object recognition (facial recognition), camera calibration, stereovision and robotic vision. This project aims to provide an easy to use and highly efficient environment. This has been achieved by its programming in optimized C and C++ code, also taking advantage of the capabilities that are provided by the multicore processors [1].

Given the great documentation and all the support from the community and the official APIs, OpenCV is ideal for projects as this one or future developments from the techniques presented in this project [2].

7.1.2. K-Means as concept testing

The project was proposed as a proof of concept, which would try to get as much information as possible, so as know the capacity of image processing applied to different fields of medicine. Are shown more examples of the solution obtained in Figures 6.1, 6.3, 6.5 . K-Means, the adopted method, has been very powerful, and could be used to identify other kinds of wounds, that is, the algorithm could be extended to other fields of medicine in which it could work equally .

7.1.3. Texture detection application to other fields

The used technique for the detection of textures, that was not absolutely positive for the proposed solution, it was noted that it may be practical in the identification in other investigations. This paper has shown the feasibility of recognizing textures through the Gabor filter which can be useful in segmentation problems.

7.2. Considerations

Throughout the project, a series of problems have emerged which are collected in this section.

7.2.1. Quality of samples

The amount of samples collected in the early stages of development were scarce, but sufficient when testing. Most of the tests were done with a few images as they were the most informative, because they had the largest number of areas to differentiate.

During development, it was noted that the image quality is a crucial factor in their treatment. Some wounds, despite high quality, did not have an acceptable level, mainly because of the lighting, excessive glare and/or focus.

In a more developed part of the investigation a number of samples were received. Notwithstanding, some images also lacked proper lighting.

A development with a greater number of pictures without quality problems would have facilitated and sped up the process, since the whole basis of the work depends on them.

7.2.2. Equalization and QP card

For proper equalization image, the QP card is included in all of them. This card is used to control the main properties of the color reproduction, the character and the temperature of it. The problem is given when the time comes to use them since we do not have the necessary application to such use.

In order to make an own image equalization, it was devised a function to find the values of the `RGB` colors. This function produced negative results, since in each corner of the card, the values of the three components which should have been exactly the same, were given absolutely different values in each of them. This caused that an algorithm for equalization of the image with the card could not be fulfilled as it would be quite an arduous task to perform.

In this way, the equalization of the image is proposed by other means eliminating the QP card in the original images to make a general equalization of them. When developing this process, the colors appeared distorted and did not approach reality causing this procedure useless.

With a precise equalization it would be possible to find out, more optimally, areas in the wounds and conclude in better results.

7.2.3. Glitters, adopted solution

The glitters' issue has been discussed for a long time by image processing experts and it is not a trivial problem. As it was explained, we achieved an algorithm able to detect glitter and delimit it inside a rectangle, so that it could get the access to the positions and approximate

sizes of the glitters points.

After several tests, it was concluded that deleting glitters efficiently was not a solution because it could not obtain the precise value of the original pixels that should be in that area of the photo, so we resolved avoiding it: When an image glitters areas are detected, two possible paths open up:

- If glitters were less than a certain percentage (a relatively small percentage, in order to use the sample) they would proceed to fill these boxes with an improbable color in a human wound of these features (green color), so if there are glitters, they would be covered by a green rectangle and obviate the execution of the `K-Means`, simply increasing the number of cores in one so all the green polygons stay in a layer apart from the rest of the information.
- Otherwise, when the brightness exceeds that percentage, the image will be discarded due to the very poor quality of the photography, as there is great part of the wound hidden by brightness as to get useful information.

In the beginning, to ensure that this mechanism works well and with the lack of algorithms that automate this process (maybe the glitters detection algorithm can detect objects that seem to be glittered, but are not), the user will be asked to specify if glitters exist or not in the image, after that it will be carried out the measurement of the percentage of those and the right decision will be taken.

7.2.4. Gabor filters

The texture detection method gave great results in a large number of images. Despite that, this method was not completely adequate for several reasons:

- The textures found in the implementation of the algorithm do not correspond to areas of *not healthy skin*. For example, in many of the photos were found hair textures areas that do not correspond to areas of bruising or granulation tissue. This could be solved with an extraction wound in its entirety.
- The problem of the photograph's brightness also affected the method of textures as excessive lighting which was also characterized as another texture. The solution to this problem was adopted in development in `K-Means` (Paragraph: 5.2.1.3).
- As each image is different, the input values for the detection of textures vary on each of the images. This means that it cannot specify default data for all images causing the modification of the introduction of such values by the user, and therefore, it is neither accurate nor reliable.

All these problems cause that the use of this procedure is not entirely applicable to the initial problem. Nevertheless, there is a possibility of extending this technique so as to be used with a higher percentage of reliability.

7.2.5. Test with K-Means.

Although the first way to use the algorithm `the K-Means` gave interesting results, its improper use had led it to invest time trying to adjust the output to get more information. But in view of the results, it was decided to pursue the investigation of this algorithm.

Instead of looking for a different embodiment for legitimate output, tests were made from the output that was obtained, which pointed out that the `the K-Means` would yield good results and these could be measured. The problem that this error brought was the absence of layers, which is what the algorithm is made to; to differ diverse layers according to a characteristic (in this case, color), so it was strange that varying the number of cores the algorithm varies a way that seemed random, though curiously whenever a value was given the picture information for that value was the same (for example, K with value of 6, the image showed areas of skin in the process healing, for K with value of 10, other information and so on).

At some point, we were only given two distinct areas and its edges through the algorithm of `Canny`, so they began to look for other ways to address the lack of layers reported by `the K -Means`. Obviously it was concluded that the use of the algorithm was not right and after a series of adjustments a good performance was achieved, along with the data we were looking for from the beginning.

This reports, still benefit to research, since the `the K-Means` is able to draw the different areas of the wound, but also found another use for details `K-Means` to differentiate good from the bad of the wound, so that, despite the small loss of time, the results do not have to be ignored.

7.2.6. Problems with the solution obtained by K-Means

Although it is the best solution found, it has some flaws that prevent it to be perfect.

7.2.6.1. Not always the dark area is not healthy skin

In section 5.2.1.2 it was thought that could be classified as healthy an area based on the darkness of the color, so it is assumed that a clear zone is healthier than dark skin area. And Figure 5.11 as an example set.

Nonetheless, as the algorithm has been tested with the samples that came later, that principle that was assumed is not always true.

In Figure 6.7 there is not only the problem of flash (already mentioned in section 5.2.1.3 and 6.2.3) which makes the glitter be identified by `K-Means` as an image color and associate it to healthy skin, but also the healthy skin is darker than the wound area.

Applying the selected reasoning, it would be making a major mistake, identifying healthy areas backwards.

7.2.6.2. Defining the clusters number, how many areas to differentiate?

It could be said that generally there are three areas to distinguish: healthy skin, epithelialization zone and granulation tissue.

However, in the new samples appear new areas that may be of sufficient importance to identify.

Not all images have the same areas to differentiate, and cannot generalize the clusters number the `K-Means`.

As a solution, user could be asked for the number of areas to differentiate.

7.2.6.3. Imperfect cropped of the wound image

The cropping of images is very important because properly insulation of the wound can make the difference between a correct processing and other that is not.

If external elements appear in the picture to be processed as a result of imperfect cropping (hospital bed sheet, patient's clothing ...), this processing will be irretrievably altered, considering that these areas will be identified in the `K-Means` as part of the wound, and alter other areas.

With a large number of samples we could look for patterns to remove certain colors, for example, remove the typical green hospital sheets and not include it in the processing.

7.3. Future work

This project has shed some light on new ways and disciplines that could take advantage of computer vision techniques to improve or streamline their work.

7.3.1. OpenCV and medicine

`OpenCV` has proven to be a compatible technology with medical applications. By a professional team and a greater amount of time, the findings would increase exponentially. It is a new field that there are still many things to discover, but the idea of the tests that have been done in this project is to speed up (never replace, since the expert opinion of a doctor weighs much more than the provision of a small software) certain tedious jobs or help in an emergency. For example, if a nursing assistant, instead of analyzing wounds one by one and reviewing records, with an application archive it could evaluate past and current statuses, it would be convenient and fast. While on the other hand, it could really help to save lives in humanitarian disasters in which every second counts, leaving software to makes the first recognition of severity to expedite the process.

7.3.2. Machine Learning

The limitations of time and material prevented further research paths that opened during the project development.

The optimal continuation of the experiment described was to apply machine learning techniques (`Machine Learning`) to acquire by, a pre-programmed algorithm, the characteristics of the image itself once the layers are separated. Furthermore, recognition of textures could have been much more powerful by these methods, but they would have taken hundreds of pictures with a certain characteristic, that would be the target characteristic, or the time needed to break the pictures into small pieces to have many small samples of healthy skin, granulocytes, bruising, etc.., which began to do, but took a long time and the lack of software that automatize made this way was abandoned despite previous research (since OpenCV has methods to train learning systems aimed at recognizing a feature, such as the detection of faces or hands, but has hundreds of thousands of samples for training, while the project did not have that much).

Referencias

[1] Opencv. http://es.wikipedia.org/wiki/OpenCV.

[2] Opencv api. http://opencv.org/.

[3] Licencia bsd. http://opensource.org/licenses/BSD-3-Clause, 1998.

[4] Filtro de gabor. http://es.wikipedia.org/wiki/Filtro_de_Gabor, 2006.

[5] Histogram equalization. http://en.wikipedia.org/wiki/Histogram_equalization, 2006.

[6] Aplicando filtros de gabor a segmentación de caracteres alfanuméricos en placas vehiculares. http://www.academia.edu/1945940/Aplicando_filtros_de_Gabor_a_segmentacion_de_caracteres_alfanumericos_en_placas_vehiculares, 2011.

[7] Basic thresholding operation. http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html, 2014.

[8] Eyal Arubas. Configuración opencv. http://eyalarubas.com/opencv-installation-on-windows-netbeans-mingw.html, 2012.

[9] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[10] J.G. Daugman. *Uncertainty relations for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters, Journal of the Optical Society of America A, vol. 2.* 1985.

[11] N. Petkov and M.B. Wieling. Servidor matlab. http://matlabserver.cs.rug.nl/edgedetectionweb/web/index.html, 2006.

[12] Simon Prince. *Computer Vision*. Cambridge University Press, 2012.